W3School Xml 教程

wizardforcel

Published with GitBook



目錄

W3School XML教程

来源：

- XML教程
- DTD教程
- XML DOM教程
- XSLT教程
- XPath教程
- XQuery教程
- XLink教程
- XPointer教程
- Schema教程
- XSL-FO教程
- SVG教程
- WSDL教程
- SOAP教程
- RSS教程
- RDF教程

整理：飞龙

XML 基础

XML 简介

XML 被设计用来传输和存储数据。

HTML 被设计用来显示数据。

应该掌握的基础知识

在您继续学习之前，需要对以下知识有基本的了解：

- HTML
- JavaScript

如果您希望首先学习这些项目，请在我们的 首页 访问这些教程。

什么是 XML？

- XML 指可扩展标记语言（EXtensible Markup Language）。
- XML 是一种很像HTML的标记语言。
- XML 的设计宗旨是传输数据，而不是显示数据。
- XML 标签没有被预定义。您需要自行定义标签。
- XML 被设计为具有自我描述性。
- XML 是 W3C 的推荐标准。

XML 和 HTML 之间的差异

XML 不是 HTML 的替代。

XML 和 HTML 为不同的目的而设计：

- XML 被设计用来传输和存储数据，其焦点是数据的内容。
- HTML 被设计用来显示数据，其焦点是数据的外观。

HTML 旨在显示信息，而 XML 旨在传输信息。

XML 不会做任何事情

也许这有点难以理解，但是 XML 不会做任何事情。XML 被设计用来结构化、存储以及传输信息。

下面实例是 Jani 写给 Tove 的便签，存储为 XML：

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

上面的这条便签具有自我描述性。它包含了发送者和接受者的信息，同时拥有标题以及消息主体。

但是，这个 XML 文档仍然没有做任何事情。它仅仅是包装在 XML 标签中的纯粹的信息。我们需要编写软件或者程序，才能传送、接收和显示出这个文档。

通过 XML 您可以发明自己的标签

上面实例中的标签没有在任何 XML 标准中定义过（比如 <to> 和 <from>）。这些标签是由 XML 文档的创作者发明的。

这是因为 XML 语言没有预定义的标签。

HTML 中使用的标签都是预定义的。HTML 文档只能使用在 HTML 标准中定义过的标签（如 <p>、<h1> 等等）。

XML 允许创作者定义自己的标签和自己的文档结构。

XML 不是对 HTML 的替代

XML 是对 **HTML** 的补充。

XML 不会替代 HTML，理解这一点很重要。在大多数 Web 应用程序中，XML 用于传输数据，而 HTML 用于格式化并显示数据。

对 XML 最好的描述是：

XML 是独立于软件和硬件的信息传输工具。

XML 是 W3C 的推荐标准

XML 于 1998 年 2 月 10 日成为 W3C 的推荐标准。

如需了解有关 W3C XML 活动的更多信息，请访问我们的 W3C 教程。

XML 无所不在

目前，XML 在 Web 中起到的作用不会亚于一直作为 Web 基石的 HTML。

XML 是各种应用程序之间进行数据传输的最常用的工具。

XML 用途

XML 应用于 Web 开发的许多方面，常用于简化数据的存储和共享。

XML 把数据从 HTML 分离

如果您需要在 HTML 文档中显示动态数据，那么每当数据改变时将花费大量的时间来编辑 HTML。

通过 XML，数据能够存储在独立的 XML 文件中。这样您就可以专注于使用 HTML/CSS 进行显示和布局，并确保修改底层数据不再需要对 HTML 进行任何的改变。

通过使用几行 JavaScript 代码，您就可以读取一个外部 XML 文件，并更新您的网页的数据内容。

XML 简化数据共享

在真实的世界中，计算机系统和数据使用不兼容的格式来存储数据。

XML 数据以纯文本格式进行存储，因此提供了一种独立于软件和硬件的数据存储方法。

这让创建不同应用程序可以共享的数据变得更加容易。

XML 简化数据传输

对开发人员来说，其中一项最费时的挑战一直是在互联网上的不兼容系统之间交换数据。

由于可以通过各种不兼容的应用程序来读取数据，以 XML 交换数据降低了这种复杂性。

XML 简化平台变更

升级到新的系统（硬件或软件平台），总是非常费时的。必须转换大量的数据，不兼容的数据经常会丢失。

XML 数据以文本格式存储。这使得 XML 在不损失数据的情况下，更容易扩展或升级到新的操作系统、新的应用程序或新的浏览器。

XML 使您的数据更有用

不同的应用程序都能够访问您的数据，不仅仅在 HTML 页中，也可以从 XML 数据源中进行访问。

通过 XML，您的数据可供各种阅读设备使用（掌上计算机、语音设备、新闻阅读器等），还可以供盲人或其他残障人士使用。

XML 用于创建新的互联网语言

很多新的互联网语言是通过 XML 创建的。

这里有一些实例：

- XHTML
- 用于描述可用的 Web 服务 的 WSDL
- 作为手持设备的标记语言的 WAP 和 WML
- 用于新闻 feed 的 RSS 语言
- 描述资本和本体的 RDF 和 OWL
- 用于描述针针对 Web 的多媒体 的 SMIL

假如开发人员都是理性的

假如他们都是理性的，就让未来的应用程序使用 **XML** 来交换数据吧。

未来也许会出现某种字处理软件、电子表格程序以及数据库，它们可以使用 XML 格式读取彼此的数据，而不需要使用任何的转换程序。

XML 树结构

XML 文档形成了一种树结构，它从"根部"开始，然后扩展到"枝叶"。

一个 XML 文档实例

XML 文档使用简单的具有自我描述性的语法：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

第一行是 XML 声明。它定义 XML 的版本（1.0）和所使用的编码（ISO-8859-1 = Latin-1/西欧字符集）。

下一行描述文档的根元素（像在说："本文档是一个便签"）：

```
<note>
```

接下来 4 行描述根的 4 个子元素（to, from, heading 以及 body）：

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

最后一行定义根元素的结尾：

```
</note>
```

您可以假设，从这个实例中，XML 文档包含了一张 Jani 写给 Tove 的便签。

XML 具有出色的自我描述性，您同意吗？

XML 文档形成一种树结构

XML 文档必须包含根元素。该元素是所有其他元素的父元素。

XML 文档中的元素形成了一棵文档树。这棵树从根部开始，并扩展到树的最底端。

所有的元素都可以有子元素：

```
<root>
<child>
<subchild>.....</subchild>
</child>
</root>
```

父、子以及同胞等术语用于描述元素之间的关系。父元素拥有子元素。相同层级上的子元素成为同胞（兄弟或姐妹）。

所有的元素都可以有文本内容和属性（类似 HTML 中）。

实例：



上图表示下面的 XML 中的一本书：

```
<bookstore>
<book category="COOKING">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
<book category="CHILDREN">
<title lang="en">Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
<book category="WEB">
<title lang="en">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```

实例中的根元素是 <bookstore>。文档中的所有 <book> 元素都被包含在 <bookstore> 中。

<book> 元素有 4 个子元素：<title>、<author>、<year>、<price>。

XML 语法规则

XML 的语法规则很简单，且很有逻辑。这些规则很容易学习，也很容易使用。

所有的 XML 元素都必须有一个关闭标签

在 HTML 中，某些元素不必有一个关闭标签：

```
<p>This is a paragraph.
<br>
```

在 XML 中，省略关闭标签是非法的。所有元素都必须有关闭标签：

```
<p>This is a paragraph.</p>
<br />
```

注释：从上面的实例中，您也许已经注意到 XML 声明没有关闭标签。这不是错误。声明不是 XML 文档本身的一部分，它没有关闭标签。

XML 标签对大小写敏感

XML 标签对大小写敏感。标签 <Letter> 与标签 <letter> 是不同的。

必须使用相同的大小写来编写打开标签和关闭标签：

```
<Message>This is incorrect</message>
<message>This is correct</message>
```

注释：打开标签和关闭标签通常被称为开始标签和结束标签。不论您喜欢哪种术语，它们的概念都是相同的。

XML 必须正确嵌套

在 HTML 中，常会看到没有正确嵌套的元素：

```
<b><i>This text is bold and italic</b></i>
```

在 XML 中，所有元素都必须彼此正确地嵌套：

```
<b><i>This text is bold and italic</i></b>
```

在上面的实例中，正确嵌套的意思是：由于 <i> 元素是在 <b> 元素内打开的，那么它必须在 <b> 元素内关闭。

XML 文档必须有根元素

XML 文档必须有一个元素是所有其他元素的父元素。该元素称为根元素。

```
<root>
<child>
<subchild>.....</subchild>
</child>
</root>
```

XML 属性值必须加引号

与 HTML 类似，XML 元素也可拥有属性（名称/值的对）。

在 XML 中，XML 的属性值必须加引号。

请研究下面的两个 XML 文档。 第一个是错误的，第二个是正确的：

```
<note date=12/11/2007>
<to>Tove</to>
<from>Jani</from>
</note>
```

```
<note date="12/11/2007">
<to>Tove</to>
<from>Jani</from>
</note>
```

在第一个文档中的错误是，note 元素中的 date 属性没有加引号。

实体引用

在 XML 中，一些字符拥有特殊的意义。

如果您把字符 "<" 放在 XML 元素中，会发生错误，这是因为解析器会把它当作新元素的开始。

这样会产生 XML 错误：

```
<message>if salary < 1000 then</message>
```

为了避免这个错误，请用实体引用来代替 "<" 字符：

```
<message>if salary &lt; 1000 then</message>
```

在 XML 中，有 5 个预定义的实体引用：

| < | < | less than | | > | > | greater than | | & | & | ampersand | | ' | ' | apostrophe | | " | " | quotation mark |

注释：在 XML 中，只有字符 "<" 和 "&" 确实是非法的。大于号是合法的，但是用实体引用来代替它是一个好习惯。

XML 中的注释

在 XML 中编写注释的语法与 HTML 的语法很相似。

<!-- This is a comment -->

在 XML 中，空格会被保留

HTML 会把多个连续的空格字符裁减（合并）为一个：

| HTML: | Hello Tove | | Output: | Hello Tove |

在 XML 中，文档中的空格不会被删减。

XML 以 LF 存储换行

在 Windows 应用程序中，换行通常以一对字符来存储：回车符（CR）和换行符（LF）。

在 Unix 和 Mac OSX 中，使用 LF 来存储新行。

在旧的 Mac 系统中，使用 CR 来存储新行。

XML 以 LF 存储换行。

XML 元素

XML 文档包含 XML 元素。

什么是 XML 元素？

XML 元素指的是从（且包括）开始标签直到（且包括）结束标签的部分。

一个元素可以包含：

- 其他元素
- 文本
- 属性
- 或混合以上所有...

```
<bookstore>
<book category="CHILDREN">
<title>Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
<book category="WEB">
<title>Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```

在上面的实例中，<bookstore> 和 <book> 都有 元素内容，因为他们包含其他元素。<book> 元素也有属性（category="CHILDREN"）。<title>、<author>、<year> 和 <price> 有文本内容，因为他们包含文本。

XML 命名规则

XML 元素必须遵循以下命名规则：

- 名称可以包含字母、数字以及其他的字符
- 名称不能以数字或者标点符号开始
- 名称不能以字母 xml（或者 XML、Xml 等等）开始
- 名称不能包含空格

可使用任何名称，没有保留的字词。

最佳命名习惯

使名称具有描述性。使用下划线的名称也很不错：<first_name>、<last_name>。

名称应简短和简单，比如：<book_title>，而不是：<the_title_of_the_book>。

避免 "-" 字符。如果您按照这样的方式进行命名："first-name"，一些软件会认为您想要从 first 里边减去 name。

避免 "." 字符。如果您按照这样的方式进行命名："first.name"，一些软件会认为 "name" 是对象 "first" 的属性。

避免 ":" 字符。冒号会被转换为命名空间来使用（稍后介绍）。

XML 文档经常有一个对应的数据库，其中的字段会对应 XML 文档中的元素。有一个实用的经验，即使用数据库的命名规则来命名 XML 文档中的元素。

在 XML 中，éòá 等非英语字母是完全合法的，不过需要留意，您的软件供应商不支持这些字符时可能出现的问题。

XML 元素是可扩展的

XML 元素是可扩展，以携带更多的信息。

请看下面的 XML 实例：

```
<note>
<to>Tove</to>
<from>Jani</from>
<body>Don't forget me this weekend!</body>
</note>
```

让我们设想一下，我们创建了一个应用程序，可将 <to>、<from> 以及 <body> 元素从 XML 文档中提取出来，并产生以下的输出：

MESSAGE

To: Tove **From:** Jani

Don't forget me this weekend!

想象一下，XML 文档的作者添加的一些额外信息：

```
<note>
<date>2008-01-10</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

那么这个应用程序会中断或崩溃吗？

不会。这个应用程序仍然可以找到 XML 文档中的 <to>、<from> 以及 <body> 元素，并产生同样的输出。

XML 的优势之一，就是可以在不中断应用程序的情况下进行扩展。

XML 属性

XML元素具有属性，类似 HTML。

属性（Attribute）提供有关元素的额外信息。

XML 属性

在 HTML 中，属性提供有关元素的额外信息：

```
<img src="computer.gif">
<a href="demo.html">
```

属性通常提供不属于数据组成部分的信息。在下面的实例中，文件类型与数据无关，但是对需要处理这个元素的软件来说却很重要：

```
<file type="gif">computer.gif</file>
```

XML 属性必须加引号

属性值必须被引号包围，不过单引号和双引号均可使用。比如一个人的性别，person 元素可以这样写：

```
<person sex="female">
```

或者这样也可以：

```
<person sex='female'>
```

如果属性值本身包含双引号，您可以使用单引号，就像这个实例：

```
<gangster name='George "Shotgun" Ziegler'>
```

或者您可以使用字符实体：

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

XML 元素 vs. 属性

请看这些实例：

```
<person sex="female">
<firstname>Anna</firstname>
<lastname>Smith</lastname>
</person>
```

```
<person>
<sex>female</sex>
<firstname>Anna</firstname>
<lastname>Smith</lastname>
</person>
```

在第一个实例中，sex 是一个属性。在第二个实例中，sex 是一个元素。这两个实例都提供相同的信息。

没有什么规矩可以告诉我们什么时候该使用属性，而什么时候该使用元素。我的经验是在 HTML 中，属性用起来很便利，但是在 XML 中，您应该尽量避免使用属性。如果信息感觉起来很像数据，那么请使用元素吧。

我最喜欢的方式

下面的三个 XML 文档包含完全相同的信息：

第一个实例中使用了 date 属性：

```
<note date="10/01/2008">
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

第二个实例中使用了 date 元素：

```
<note>
<date>10/01/2008</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

第三个实例中使用了扩展的 date 元素（这是我的最爱）：

```
<note>
<date>
<day>10</day>
<month>01</month>
<year>2008</year>
</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

避免 XML 属性？

因使用属性而引起的一些问题：

- 属性不能包含多个值（元素可以）
- 属性不能包含树结构（元素可以）
- 属性不容易扩展（为未来的变化）

属性难以阅读和维护。请尽量使用元素来描述数据。而仅仅使用属性来提供与数据无关的信息。

不要做这样的蠢事（这不是 XML 应该被使用的方式）：

```
<note day="10" month="01" year="2008"
to="Tove" from="Jani" heading="Reminder"
body="Don't forget me this weekend!">
</note>
```

针对元数据的 XML 属性

有时候会向元素分配 ID 引用。这些 ID 索引可用于标识 XML 元素，它起作用的方式与 HTML 中 id 属性是一样的。这个实例向我们演示了这种情况：

```
<messages>
<note id="501">
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
<note id="502">
<to>Jani</to>
<from>Tove</from>
<heading>Re: Reminder</heading>
<body>I will not</body>
</note>
</messages>
```

上面的 id 属性仅仅是一个标识符，用于标识不同的便签。它并不是便签数据的组成部分。

在此我们极力向您传递的理念是：元数据（有关数据的数据）应当存储为属性，而数据本身应当存储为元素。

XML 验证

拥有正确语法的 XML 被称为"形式良好"的 XML。

通过 DTD 验证的XML是"合法"的 XML。

形式良好的 XML 文档

"形式良好"的 XML 文档拥有正确的语法。

在前面的章节描述的语法规则：

- XML 文档必须有一个根元素
- XML元素都必须有一个关闭标签
- XML 标签对大小写敏感
- XML 元素必须被正确的嵌套
- XML 属性值必须加引号

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

验证 XML 文档

合法的 XML 文档是"形式良好"的 XML 文档，这也符合文档类型定义（DTD）的规则：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

在上面的实例中，DOCTYPE 声明是对外部 DTD 文件的引用。下面的段落展示了这个文件的内容。

XML DTD

DTD 的目的是定义 XML 文档的结构。它使用一系列合法的元素来定义文档结构：

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

如果您想要学习 DTD，请在我们的首页查找 DTD 教程。

XML Schema

W3C 支持一种基于 XML 的 DTD 代替者，它名为 XML Schema：

```
<xs:element name="note">

<xs:complexType>
<xs:sequence>
<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
<xs:element name="heading" type="xs:string"/>
<xs:element name="body" type="xs:string"/>
</xs:sequence>
</xs:complexType>

</xs:element>
```

如果您想要学习 XML Schema，请在我们的首页查找 Schema 教程。

一个通用的 XML 验证器

为了帮助您检查 XML 文件的语法，我们创建了 XML 验证器，以便您对任何 XML 文件进行语法检查。

请看下一章。

查看 XML 文件

在所有主流的浏览器中，均能够查看原始的 XML 文件。

不要指望 XML 文件会直接显示为 HTML 页面。

查看 XML 文件

| <?xml version="1.0" encoding="ISO-8859-1"?>- <note><to>Tove</to><from>Jani</from><heading>Reminder</heading><body>Don't forget me this weekend!</body></note> |

查看这个 XML 文件：note.xml

XML 文档将显示为代码颜色化的根以及子元素。通过点击元素左侧的加号（+）或减号（-），可以展开或收起元素的结构。要查看原始的 XML 源（不包括 + 和 - 符号），选择"查看页面源代码"或从浏览器菜单"查看源文件"。

注释：在 Safari 中，只有元素的文本将被显示。要查看原始的 XML，您必须右键单击页面，选择"查看源文件"。

查看无效的 XML 文件

如果一个错误的XML文件被打开，浏览器会报告错误。

请查看这个 XML 文件：note_error.xml

其他 XML 实例

请查看这些 XML 文档，这会有助于您建立对 XML 的感性认识。

一个 XML 的 CD 目录 这是一个 CD 集，存储为 XML 数据。

一个 XML 的植物目录 这是一个来自植物店的植物目录，存储为 XML 数据。

一个简单的食物菜单 这是一个来自餐馆的早餐菜单，存储为 XML 数据。

为什么 XML 显示这个样子？

XML 文档不会携带有关如何显示数据的信息。

由于 XML 标签由 XML 文档的作者"发明"，浏览器无法确定像 <table> 这样一个标签究竟描述一个 HTML 表格还是一个餐桌。

在没有任何有关如何显示数据的信息的情况下，大多数的浏览器都会仅仅把 XML 文档显示为源代码。

在下面的章节，我们会了解几个有关这个显示问题的解决方案，其中会使用 CSS、XSLT 和 JavaScript。

使用 CSS 显示 XML

通过使用 CSS（Cascading Style Sheets 层叠样式表），您可以添加显示信息到 XML 文档中。

使用 CSS 显示您的 XML？

使用 CSS 来格式化 XML 文档是有可能的。

下面的实例就是关于如何使用 CSS 样式表来格式化 XML 文档：

请看这个 XML 文件：CD 目录

然后看这个样式表：CSS 文件

最后，请查看：使用 CSS 文件格式化的 CD 目录

下面是 XML 文件的一小部分。第二行把 XML 文件链接到 CSS 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Hide your heart</TITLE>
<ARTIST>Bonnie Tyler</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
.
.
.
</CATALOG>
```

使用 CSS 格式化 XML 不是常用的方法。

W3C 推荐使用 XSLT，请看下一章。

使用 XSLT 显示 XML

通过使用 XSLT，您可以把 XML 文档转换成 HTML 格式。

使用 XSLT 显示 XML

XSLT 是首选的 XML 样式表语言。

XSLT（eXtensible Stylesheet Language Transformations）远比 CSS 更加完善。

XSLT 是在浏览器显示 XML 文件之前，先把它转换为 HTML：

使用 XSLT 显示 XML

如果您想要学习有关 XSLT 的知识，请在我们的首页查找 XSLT 教程。

在服务器上通过 XSLT 转换 XML

在上面的实例中，当浏览器读取 XML 文件时，XSLT 转换是由浏览器完成的。

在使用 XSLT 来转换 XML 时，不同的浏览器可能会产生不同结果。为了减少这种问题，可以在服务器上进行 XSLT 转换。

查看结果。

XML Javascript

XMLHttpRequest 对象

XMLHttpRequest 对象

XMLHttpRequest 对象用于在后台与服务器交换数据。

XMLHttpRequest 对象是开发者的梦想，因为您能够：

- 在不重新加载页面的情况下更新网页
- 在页面已加载后从服务器请求数据
- 在页面已加载后从服务器接收数据
- 在后台向服务器发送数据

如需学习更多关于 XMLHttpRequest 对象的知识，请学习我们的 XML DOM 教程。

XMLHttpRequest 实例

当你在下面的输入字段中键入一个字符，一个 XMLHttpRequest 发送到服务器 - 返回名称的建议（从服务器上的文件）：

在输入框中键入一个字母：

```
首字母

<input type="text" id="txt1" onkeyup="showHint(this.value)">

建议:
```

创建一个 XMLHttpRequest 对象

所有现代浏览器（IE7+、Firefox、Chrome、Safari 和 Opera）都有内建的 XMLHttpRequest 对象。

创建 XMLHttpRequest 对象的语法：

```
xmlhttp=new XMLHttpRequest();
```

旧版本的Internet Explorer（IE5和IE6）中使用 ActiveX 对象：

```
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
```

在下一章中，我们将使用 XMLHttpRequest 对象从服务器取回 XML 信息。

XML Parser

所有现代浏览器都有内建的 XML 解析器。

XML 解析器把 XML 文档转换为 XML DOM 对象 - 可通过 JavaScript 操作的对象。

解析 XML 文档

下面的代码片段把 XML 文档解析到 XML DOM 对象中：

```
if (window.XMLHttpRequest)
{// code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp=new XMLHttpRequest();
}
else
{// code for IE6, IE5
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.open("GET","books.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;
```

解析 XML 字符串

下面的代码片段把 XML 字符串解析到 XML DOM 对象中：

```
txt="<bookstore><book>";
txt=txt+"<title>Everyday Italian</title>";
txt=txt+"<author>Giada De Laurentiis</author>";
txt=txt+"<year>2005</year>";
txt=txt+"</book></bookstore>";

if (window.DOMParser)
{
parser=new DOMParser();
xmlDoc=parser.parseFromString(txt,"text/xml");
}
else // Internet Explorer
{
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.loadXML(txt);
}
```

注释：Internet Explorer 使用 loadXML() 方法来解析 XML 字符串，而其他浏览器使用 DOMParser 对象。

跨域访问

出于安全方面的原因，现代的浏览器不允许跨域的访问。

这意味着，网页以及它试图加载的 XML 文件，都必须位于相同的服务器上。

XML DOM

在下一章中，您将学习如何访问 XML DOM 对象并取回数据。

XML DOM

DOM（Document Object Model 文档对象模型）定义了访问和操作文档的标准方法。

XML DOM

XML DOM（XML Document Object Model）定义了访问和操作 XML 文档的标准方法。

XML DOM 把 XML 文档作为树结构来查看。

所有元素可以通过 DOM 树来访问。可以修改或删除它们的内容，并创建新的元素。元素，它们的文本，以及它们的属性，都被认为是节点。

在我们的 XML DOM 教程中，您可以学习更多有关 XML DOM 的知识。

HTML DOM

HTML DOM 定义了访问和操作 HTML 文档的标准方法。

所有 HTML 元素可以通过 HTML DOM 来访问。

在我们的 HTML DOM 教程中，您可以学习更多有关 HTML DOM 的知识。.

加载一个 XML 文件 - 跨浏览器实例

下面的实例把 XML 文档（"note.xml"）解析到 XML DOM 对象中，然后通过 JavaScript 提取一些信息：

实例

```
<html>
<body>
<h1>W3Schools Internal Note</h1>
<div>
<b>To:</b> <span id="to"></span><br />
<b>From:</b> <span id="from"></span><br />
<b>Message:</b> <span id="message"></span>
</div>

<script>
if (window.XMLHttpRequest)
{// code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp=new XMLHttpRequest();
}
else
{// code for IE6, IE5
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.open("GET","note.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

document.getElementById("to").innerHTML=
xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue;
document.getElementById("from").innerHTML=
xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;
document.getElementById("message").innerHTML=
xmlDoc.getElementsByTagName("body")[0].childNodes[0].nodeValue;
</script>

</body>
</html>
```

重要注释！

如需从上面的 XML 文件（"note.xml"）的 <to> 元素中提取文本 "Tove"，语法是：

```
getElementsByTagName("to")[0].childNodes[0].nodeValue
```

请注意，即使 XML 文件只包含一个 <to> 元素，您仍然必须指定数组索引 [0]。这是因为 getElementsByTagName() 方法返回一个数组。

加载一个 XML 字符串 - 跨浏览器实例

下面的实例把 XML 字符串解析到 XML DOM 对象中，然后通过 JavaScript 提取一些信息：

实例

```
<html>
<body>
<h1>W3Schools Internal Note</h1>
<div>
<b>To:</b> <span id="to"></span><br />
<b>From:</b> <span id="from"></span><br />
<b>Message:</b> <span id="message"></span>
</div>

<script>
txt="<note>";
txt=txt+"<to>Tove</to>";
txt=txt+"<from>Jani</from>";
txt=txt+"<heading>Reminder</heading>";
txt=txt+"<body>Don't forget me this weekend!</body>";
txt=txt+"</note>";

if (window.DOMParser)
{
parser=new DOMParser();
xmlDoc=parser.parseFromString(txt,"text/xml");
}
else // Internet Explorer
{
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.loadXML(txt);
}

document.getElementById("to").innerHTML=
xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue;
document.getElementById("from").innerHTML=
xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;
document.getElementById("message").innerHTML=
xmlDoc.getElementsByTagName("body")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

XML to HTML

在 HTML 页面中显示 XML 数据

在下面的实例中，我们打开一个 XML 文件（"cd_catalog.xml"），然后遍历每个 CD 元素，并显示HTML 表格中的 ARTIST 元素和 TITLE 元素的值：

实例

```
<html>
<body>

<script>
if (window.XMLHttpRequest)
{// code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp=new XMLHttpRequest();
}
else
{// code for IE6, IE5
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.open("GET","cd_catalog.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

document.write("<table border='1'>");
var x=xmlDoc.getElementsByTagName("CD");
for (i=0;i<x.length;i++)
{
document.write("<tr><td>");
document.write(x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
document.write("</td><td>");
document.write(x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
document.write("</td></tr>");
}
document.write("</table>");
</script>

</body>
</html>
```

如需了解更多关于使用 JavaScript 和 XML DOM 的信息，请访问我们的 XML DOM 教程。

XML 应用程序

本章演示一些基于 XML, HTML, XML DOM 和 JavaScript 构建的小型 XML 应用程序。

XML 文档实例

在本应用程序中，我们将使用 "cd_catalog.xml" 文件。

在 HTML div 元素中显示第一个 CD

下面的实例从第一个 CD 元素中获取 XML 数据，然后在 id="showCD" 的 HTML 元素中显示数据。displayCD() 函数在页面加载时调用：

实例

```
x=xmlDoc.getElementsByTagName("CD");
i=0;

function displayCD()
{
artist=(x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
title=(x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
year=(x[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue);
txt="Artist: " + artist + "<br />Title: " + title + "<br />Year: "+ year;
document.getElementById("showCD").innerHTML=txt;
}
```

添加导航脚本

为了向上面的实例添加导航（功能），需要创建 next() 和 previous() 两个函数：

实例

```
function next()
{ // display the next CD, unless you are on the last CD
if (i<x.length-1)
{
i++;
displayCD();
}
}

function previous()
{ // displays the previous CD, unless you are on the first CD
if (i>0)
{
i--;
displayCD();
}
}
```

当点击 CD 时显示专辑信息

最后的实例展示如何在用户点击某个 CD 项目时显示专辑信息：

尝试一下。

如需了解更多关于使用 JavaScript 和 XML DOM 的信息，请访问我们的 XML DOM 教程。

XML 进阶

XML 命名空间

XML 命名空间提供避免元素命名冲突的方法。

命名冲突

在 XML 中，元素名称是由开发者定义的，当两个不同的文档使用相同的元素名时，就会发生命名冲突。

这个 XML 携带 HTML 表格的信息：

```
<table>
<tr>
<td>Apples</td>
<td>Bananas</td>
</tr>
</table>
```

这个 XML 文档携带有关桌子的信息（一件家具）：

```
<table>
<name>African Coffee Table</name>
<width>80</width>
<length>120</length>
</table>
```

假如这两个 XML 文档被一起使用，由于两个文档都包含带有不同内容和定义的 <table> 元素，就会发生命名冲突。

XML 解析器无法确定如何处理这类冲突。

使用前缀来避免命名冲突

在 XML 中的命名冲突可以通过使用名称前缀从而容易地避免。

该 XML 携带某个 HTML 表格和某件家具的信息：

```
<h:table>
<h:tr>
<h:td>Apples</h:td>
<h:td>Bananas</h:td>
</h:tr>
</h:table>

<f:table>
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>
```

在上面的实例中，不会有冲突，因为两个 <table> 元素有不同的名称。

XML 命名空间 - xmlns 属性

当在 XML 中使用前缀时，一个所谓的用于前缀的命名空间必须被定义。

命名空间是在元素的开始标签的 **xmlns** 属性中定义的。

命名空间声明的语法如下。xmlns:前缀="*URI*"。

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
<h:tr>
<h:td>Apples</h:td>
<h:td>Bananas</h:td>
</h:tr>
</h:table>

<f:table xmlns:f="http://www.w3cschool.cc/furniture">
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>

</root>
```

在上面的实例中，<table> 标签的 xmlns 属性定义了 h: 和 f: 前缀的合格命名空间。

当命名空间被定义在元素的开始标签中时，所有带有相同前缀的子元素都会与同一个命名空间相关联。

命名空间，可以在他们被使用的元素中或者在 XML 根元素中声明：

```
<root xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3cschool.cc/furniture">

<h:table>
<h:tr>
<h:td>Apples</h:td>
<h:td>Bananas</h:td>
</h:tr>
</h:table>

<f:table>
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>

</root>
```

注释：命名空间 URI 不会被解析器用于查找信息。

其目的是赋予命名空间一个惟一的名称。不过，很多公司常常会作为指针来使用命名空间指向实际存在的网页，这个网页包含关于命名空间的信息。

请访问 http://www.w3.org/TR/html4/。

统一资源标识符（URI，全称 Uniform Resource Identifier）

统一资源标识符（URI）是一串可以标识因特网资源的字符。

最常用的 URI 是用来标识因特网域名地址的统一资源定位器（URL）。另一个不那么常用的 URI 是统一资源命名（URN）。

在我们的实例中，我们仅使用 URL。

默认的命名空间

为元素定义默认的命名空间可以让我们省去在所有的子元素中使用前缀的工作。它的语法如下：

```
xmlns="_namespaceURI_"
```

这个 XML 携带 HTML 表格的信息：

```
<table xmlns="http://www.w3.org/TR/html4/">
<tr>
<td>Apples</td>
<td>Bananas</td>
</tr>
</table>
```

这个XML携带有关一件家具的信息：

```
<table xmlns="http://www.w3schools.com/furniture">
<name>African Coffee Table</name>
<width>80</width>
<length>120</length>
</table>
```

实际使用中的命名空间

XSLT 是一种用于把 XML 文档转换为其他格式的 XML 语言，比如 HTML。

在下面的 XSLT 文档中，您可以看到，大多数的标签是 HTML 标签。

非 HTML 的标签都有前缀 xsl，并由此命名空间标识：
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr>
<th align="left">Title</th>
<th align="left">Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

如果您想要学习有关 XSLT 的知识，请在我们的首页查找 XSLT 教程。

XML CDATA

XML 文档中的所有文本均会被解析器解析。

只有 CDATA 区段中的文本会被解析器忽略。

PCDATA - 被解析的字符数据

XML 解析器通常会解析 XML 文档中所有的文本。

当某个 XML 元素被解析时，其标签之间的文本也会被解析：

```
<message>This text is also parsed</message>
```

解析器之所以这么做是因为 XML 元素可包含其他元素，就像这个实例中，其中的 <name> 元素包含着另外的两个元素（first 和 last）：

```
<name><first>Bill</first><last>Gates</last></name>
```

而解析器会把它分解为像这样的子元素：

```
<name>
<first>Bill</first>
<last>Gates</last>
</name>
```

解析字符数据（PCDATA）是 XML 解析器解析的文本数据使用的一个术语。

CDATA - （未解析）字符数据

术语 CDATA 是不应该由 XML 解析器解析的文本数据。

像 "<" 和 "&" 字符在 XML 元素中都是非法的。

"<" 会产生错误，因为解析器会把该字符解释为新元素的开始。

"&" 会产生错误，因为解析器会把该字符解释为字符实体的开始。

某些文本，比如 JavaScript 代码，包含大量 "<" 或 "&" 字符。为了避免错误，可以将脚本代码定义为 CDATA。

CDATA 部分中的所有内容都会被解析器忽略。

CDATA 部分由 "**<![CDATA[**" 开始，由 "**]]>**" 结束：

```
<script>
<![CDATA[
function matchwo(a,b)
{
if (a < b && a < 0) then
{
return 1;
}
else
{
return 0;
}
}
]]>
</script>
```

在上面的实例中，解析器会忽略 CDATA 部分中的所有内容。

关于 **CDATA** 部分的注释：

CDATA 部分不能包含字符串 "]]>"。也不允许嵌套的 CDATA 部分。

标记 CDATA 部分结尾的 "]]>" 不能包含空格或换行。

XML 编码

XML 文档可以包含非 ASCII 字符，比如挪威语 ? ? ?，或者法语 ê è é。

为了避免错误，需要规定 XML 编码，或者将 XML 文件存为 Unicode。

XML 编码错误

如果您载入一个 XML 文档，您可以得到两个不同的错误，表示编码问题：

在文本内容中发现无效字符。

如果您的 XML 中包含非 ASCII 字符，且文件保存为没有指定编码的单字节 ANSI（或 ASCII），您会得到一个错误。

单字节编码属性的 XML 文件。

相同的单字节没有编码属性的 XML 文件。

将当前编码切换为不被支持的指定编码

如果您的 XML 文件保存为带有指定的单字节编码（WINDOWS-1252、ISO-8859-1、UTF-8）的双字节 Unicode（或 UTF-16），您会得到一个错误。

如果您的 XML 文件保存为带有指定的双字节编码（UTF-16）的单字节 ANSI（或 ASCII），您也会得到一个错误。

双字节没有编码的 XML 文件。

相同的双字节具有单字节编码的 XML 文件。

Windows 记事本

Windows 记事本默认会将文件保存为单字节的 ANSI（ASCII）。

如果您选择 "另存为..."，就可以指定 ANSI、UTF-8、Unicode（UTF-16）或 Unicode Big。

将下面的 XML 保存为 ANSI、UTF-8 和 Unicode（注意文档不包含任何编码属性）。

```
<?xml version="1.0"?>
<note>
<from>Jani</from>
<to>Tove</to>
<message>Norwegian: ???. French: êèé</message>
</note>
```

尝试将文件拖到您的浏览器，并查看结果。不同的浏览器会显示不同的结果。

不同编码的体验：

```
<?xml version="1.0" encoding="us-ascii"?>
<?xml version="1.0" encoding="windows-1252"?>
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-16"?>
```

请尝试：

带有正确编码的保存

带有错误编码的保存

结论

- 始终使用编码属性
- 使用支持编码的编辑器
- 确保您知道编辑器使用什么编码
- 在您的编码属性中使用相同的编码

服务器上的 XML

XML 文件是类似 HTML 文件的纯文本文件。

XML 能够通过标准的 Web 服务器轻松地存储和生成。

在服务器上存储 XML 文件

XML 文件在 Internet 服务器上进行存储的方式与 HTML 文件完全相同。

启动 Windows 记事本，并写入以下行：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<from>Jani</from>
<to>Tove</to>
<message>Remember me this weekend</message>
</note>
```

然后用适当的文件名，比如 "note.xml"，在 Web 服务器上保存这个文件。

通过 ASP 生成 XML

XML 可在不安装任何 XML 软件的情况下在服务器端生成。

如需从服务器生成 XML 响应 - 只需简单地编写以下代码并在 Web 服务器上把它保存为一个 ASP 文件：

```
<%
response.ContentType="text/xml"
response.Write("<?xml version='1.0' encoding='ISO-8859-1'?>")
response.Write("<note>")
response.Write("<from>Jani</from>")
response.Write("<to>Tove</to>")
response.Write("<message>Remember me this weekend</message>")
response.Write("</note>")
%>
```

请注意，此响应的内容类型必须设置为 "text/xml"。

查看这个 ASP 文件如何从服务器返回。

如果您想要学习 ASP，请在我们的首页查找 ASP 教程。

通过 PHP 生成 XML

如需使用 PHP 从服务器上生成 XML 响应，请使用下面的代码：

```
<?php
header("Content-type: text/xml");
echo "<?xml version='1.0' encoding='ISO-8859-1'?>";
echo "<note>";
echo "<from>Jani</from>";
echo "<to>Tove</to>";
echo "<message>Remember me this weekend</message>";
echo "</note>";
?>
```

请注意，响应头部的内容类型必须设置为 "text/xml"。

查看这个 PHP 文件如何从服务器返回。

如果您想要学习 PHP，请在我们的首页查找 PHP 教程。

从数据库生成 XML

XML 可在不安装任何 XML 软件的情况下从数据库生成。

如需从服务器生成 XML 数据库响应，只需简单地编写以下代码，并把它在 Web 服务器上保存为 ASP 文件：

```
<%
response.ContentType = "text/xml"
set conn=Server.CreateObject("ADODB.Connection")
conn.provider="Microsoft.Jet.OLEDB.4.0;"
conn.open server.mappath("/db/database.mdb")

sql="select fname,lname from tblGuestBook"
set rs=Conn.Execute(sql)

response.write("<?xml version='1.0' encoding='ISO-8859-1'?>")
response.write("<guestbook>")
while (not rs.EOF)
response.write("<guest>")
response.write("<fname>" & rs("fname") & "</fname>")
response.write("<lname>" & rs("lname") & "</lname>")
response.write("</guest>")
rs.MoveNext()
wend

rs.close()
conn.close()
response.write("</guestbook>")
%>
```

查看以上 ASP 文件的实际数据库输出。

上面的实例使用了带有 ADO 的 ASP。

如果您想要学习 ASP 和 ADO，请在我们的首页查找相关教程。

在服务器上通过 XSLT 转换 XML

下面的 ASP 代码在服务器上把 XML 文件转换为 XHTML：

```
<%
'Load XML
set xml = Server.CreateObject("Microsoft.XMLDOM")
xml.async = false
xml.load(Server.MapPath("simple.xml"))

'Load XSL
set xsl = Server.CreateObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load(Server.MapPath("simple.xsl"))

'Transform file
Response.Write(xml.transformNode(xsl))
%>
```

实例解释

- 第一个代码块创建微软 XML 解析器的实例（XMLDOM），并把 XML 文件载入内存。
- 第二个代码块创建解析器的另一个实例，并把 XSL 文件载入内存。
- 最后一个代码使用 XSL 文档来转换 XML 文档，并把结果以 XHTML 发送到您的浏览器。

看看上面的代码怎么运行。

通过 ASP 把 XML 保存为文件

这个 ASP 实例会创建一个简单的 XML 文档，并把该文档保存到服务器上：

```
<%
text="<note>"
text=text & "<to>Tove</to>"
text=text & "<from>Jani</from>"
text=text & "<heading>Reminder</heading>"
text=text & "<body>Don't forget me this weekend!</body>"
text=text & "</note>"

set xmlDoc=Server.CreateObject("Microsoft.XMLDOM")
xmlDoc.async=false
xmlDoc.loadXML(text)

xmlDoc.Save("test.xml")
%>
```

XML DOM 高级

XML DOM - 高级

在本教程的较早章节中，我们介绍了 XML DOM，并使用了 XML DOM 的 getElementsByTagName() 方法从 XML 文档中取回数据。

在本章中我们将结石一些其他重要的 XML DOM 方法。

您可以在我们的 XML DOM 教程 中学习更多有关 XML DOM 的知识。

获取元素的值

下面的实例中使用的 XML 文件：books.xml。

下面的实例检索第一个 <title> 元素的文本值：

实例

```
txt=xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
```

获取属性的值

下面的实例检索第一个 <title> 元素的 "lang" 属性的文本值：

实例

```
txt=xmlDoc.getElementsByTagName("title")[0].getAttribute("lang");
```

改变元素的值

下面的实例改变第一个 <title> 元素的文本值：

实例

```
x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.nodeValue="Easy Cooking";
```

创建新的属性

XML DOM 的 setAttribute() 方法可用于改变现有的属性值，或创建一个新的属性。

下面的实例创建了一个新的属性（edition="first"），然后把它添加到每一个 <book> 元素中：

实例

```
x=xmlDoc.getElementsByTagName("book");

for(i=0;i<x.length;i++)
{
x[i].setAttribute("edition","first");
}
```

创建元素

XML DOM 的 createElement() 方法创建一个新的元素节点。

XML DOM 的 createTextNode() 方法创建一个新的文本节点。

XML DOM 的 appendChild() 方法向节点添加子节点（在最后一个子节点之后）。

如需创建带有文本内容的新元素，需要同时创建元一个新的元素节点和一个新的文本节点，然后把他追加到现有的节点。

下面的实例创建了一个新的元素（<edition>），带有如下文本：First，然后把它添加到第一个 <book> 元素：

实例

```
newel=xmlDoc.createElement("edition");
newtext=xmlDoc.createTextNode("First");
newel.appendChild(newtext);

x=xmlDoc.getElementsByTagName("book");
x[0].appendChild(newel);
```

实例解释

- 创建一个 <edition> 元素
- 创建值为 "First" 的文本节点

- 把这个文本节点追加到新的 <edition> 元素
- 把 <edition> 元素追加到第一个 <book> 元素

删除元素

下面的实例删除第一个 <book> 元素的第一个节点：

实例

```
x=xmlDoc.getElementsByTagName("book")[0];
x.removeChild(x.childNodes[0]);
```

注释：上面实例的结果可能会根据所用的浏览器而不同。Firefox 把新行字符当作空的文本节点，而 Internet Explorer 不是这样。您可以在我们的 XML DOM 教程 中阅读到更多有关这个问题以及如何避免它的知识。

XML 注意事项

这里列出了您在使用 XML 时应该尽量避免使用的技术。

Internet Explorer - XML 数据岛

它是什么？XML 数据岛是嵌入到 HTML 页面中的 XML 数据。

为什么要避免使用它？XML 数据岛只在 Internet Explorer 浏览器中有效。

用什么代替它？您应当在 HTML 中使用 JavaScript 和 XML DOM 来解析并显示 XML。

如需更多有关 JavaScript 和 XML DOM 的信息，请访问我们的 XML DOM 教程。

XML 数据岛实例

本例使用 XML 文档 "cd_catalog.xml"。

把 XML 文档绑定到 HTML 文档中的一个 <xml> 标签。id 属性定义数据岛的标识符，而 src 属性指向 XML 文件：

实例

本实例只适用于 IE 浏览器

```
<html>
<body>

<xml id="cdcat" src="cd_catalog.xml"></xml>

<table border="1" datasrc="#cdcat">
<tr>
<td><span datafld="ARTIST"></span></td>
<td><span datafld="TITLE"></span></td>
</tr>
</table>

</body>
</html>
```

<table> 标签的 datasrc 属性把 HTML 表格绑定到 XML 数据岛。

<span> 标签允许 datafld 属性引用要显示的 XML 元素。在这个实例中，要引用的是 "ARTIST" 和 "TITLE"。当读取 XML 时，会为每个 <CD> 元素创建相应的表格行。

Internet Explorer - 行为

它是什么？Internet Explorer 5 引入了行为。行为是通过使用 CSS 样式向 XML （或 HTML ）元素添加行为的一种方法。

为什么要避免使用它？只有 Internet Explorer 支持 behavior 属性。

使用什么代替它？使用 JavaScript 和 XML DOM（或 HTML DOM）来代替它。

实例 1 - 鼠标悬停突出

下面的 HTML 文件中的 <style> 元素为 <h1> 元素定义了一个行为：

```
<html>
<head>
<style type="text/css">
h1 { behavior: url(behave.htc) }
</style>
</head>
<body>

<h1>Mouse over me!!!</h1>

</body>
</html>
```

下面显示的是 XML 文档 "behave.htc"（该文件包含了一段 JavaScript 和针对元素的事件句柄）：

```
<attach for="element" event="onmouseover" handler="hig_lite" />
<attach for="element" event="onmouseout" handler="low_lite" />

<script>
function hig_lite()
{
element.style.color='red';
}

function low_lite()
{
element.style.color='blue';
}
</script>
```

实例 2 - 打字机模拟

下面的 HTML 文件中的 <style> 元素为 id 为 "typing" 的元素定义了一个行为：

```
<html>
<head>
<style type="text/css">
#typing
{
behavior:url(typing.htc);
font-family:'courier new';
}
</style>
</head>
<body>

<span id="typing" speed="100">IE5 introduced DHTML behaviors.
Behaviors are a way to add DHTML functionality to HTML elements
with the ease of CSS.<br /><br />How do behaviors work?<br />
By using XML we can link behaviors to any element in a web page
and manipulate that element.</p>v </span>

</body>
</html>
```

下面显示的是 XML 文档 "typing.htc"：

```
<attach for="window" event="onload" handler="beginTyping" />
<method name="type" />

<script>
var i,text1,text2,textLength,t;

function beginTyping()
{
i=0;
text1=element.innerText;
textLength=text1.length;
element.innerText="";
text2="";
t=window.setInterval(element.id+".type()",speed);
}

function type()
{
text2=text2+text1.substring(i,i+1);
element.innerText=text2;
i=i+1;
if (i==textLength)
{
clearInterval(t);
}
}
</script>
```

XML 相关技术

下面是一个 XML 技术的列表。

XHTML (可扩展 HTML) 更严格更纯净的基于 XML 的 HTML 版本。

XML DOM (XML 文档对象模型) 访问和操作 XML 的标准文档模型。

XSL (可扩展样式表语言) XSL 包含三个部分：

- XSLT (XSL 转换) - 把 XML 转换为其他格式，比如 HTML
- XSL-FO (XSL 格式化对象)- 用于格式化 XML 文档的语言
- XPath - 用于导航 XML 文档的语言

XQuery (XML 查询语言) 基于 XML 的用于查询 XML 数据的语言。

DTD (文档类型定义) 用于定义 XML 文档中的合法元素的标准。

XSD (XML 架构) 基于 XML 的 DTD 替代物。

XLink (XML 链接语言) 在 XML 文档中创建超级链接的语言。

XPointer (XML 指针语言) 允许 XLink 超级链接指向 XML 文档中更多具体的部分。

SOAP (简单对象访问协议) 允许应用程序在 HTTP 之上交换信息的基于 XML 的协议。

WSDL (Web 服务描述语言) 用于描述网络服务的基于 XML 的语言。

RDF (资源描述框架) 用于描述网络资源的基于 XML 的语言。

RSS (真正简易聚合) 聚合新闻以及类新闻站点内容的格式。

SVG (可伸缩矢量图形) 定义 XML 格式的图形。

现实生活中的 XML

如何使用 XML 来交换信息的一些实例。

实例：XML 新闻

XMLNews 是用于交换新闻和其他信息的规范。

对新闻的供求双方来说，通过使用这种标准，可以使各种类型的新闻信息通过不同软硬件以及编程语言进行的制作、接收和存档更加容易：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nitf>
<head>
<title>Colombia Earthquake</title>
</head>
<body>
<headline>
<hl1>143 Dead in Colombia Earthquake</hl1>
</headline>
<byline>
<bytag>By Jared Kotler, Associated Press Writer</bytag>
</byline>
<dateline>
<location>Bogota, Colombia</location>
<date>Monday January 25 1999 7:28 ET</date>
</dateline>
</body>
</nitf>
```

实例：XML 气象服务

XML 国家气象服务案例，来自 NOAA（National Oceanic and Atmospheric Administration）：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<current_observation>

<credit>NOAA's National Weather Service</credit>
<credit_URL>http://weather.gov/</credit_URL>

<image>
<url>http://weather.gov/images/xml_logo.gif</url>
<title>NOAA's National Weather Service</title>
<link>http://weather.gov</link>
</image>

<location>New York/John F. Kennedy Intl Airport, NY</location>
<station_id>KJFK</station_id>
<latitude>40.66</latitude>
<longitude>-73.78</longitude>
<observation_time_rfc822>Mon, 11 Feb 2008 06:51:00 -0500 EST
</observation_time_rfc822>

<weather>A Few Clouds</weather>
<temp_f>11</temp_f>
<temp_c>-12</temp_c>
<relative_humidity>36</relative_humidity>
<wind_dir>West</wind_dir>
<wind_degrees>280</wind_degrees>
<wind_mph>18.4</wind_mph>
<wind_gust_mph>29</wind_gust_mph>
<pressure_mb>1023.6</pressure_mb>
<pressure_in>30.23</pressure_in>
<dewpoint_f>-11</dewpoint_f>
<dewpoint_c>-24</dewpoint_c>
<windchill_f>-7</windchill_f>
<windchill_c>-22</windchill_c>
<visibility_mi>10.00</visibility_mi>

<icon_url_base>http://weather.gov/weather/images/fcicons/</icon_url_base>
<icon_url_name>nfew.jpg</icon_url_name>
<disclaimer_url>http://weather.gov/disclaimer.html</disclaimer_url>
<copyright_url>http://weather.gov/disclaimer.html</copyright_url>

</current_observation>
```

XML 编辑器

如果您希望极认真地学习和使用 XML，那么您一定会从一款专业的 XML 编辑器的使用上受益。

XML 是基于文本的

XML 是基于文本的标记语言。

关于 XML 的一件很重要的事情是，XML 可被类似记事本这样的简单的文本编辑器来创建和编辑。

不过，在您开始使用 XML 进行工作时，您很快会发现，使用一款专业的 XML 编辑器来编辑 XML 文档会更好。

为什么不使用记事本?

许多 Web 开发人员使用记事本来编辑 HTML 和 XML 文档，这是因为最常用的操作系统都带有记事本，而且它很容易使用。从个人来讲，我经常使用记事本来快速地编辑某些简单的 HTML、CSS 以及 XML 文件。

但是，如果您将记事本用于 XML 编辑，可能很快会发现不少问题。

记事本不能确定您编辑的文档类型，所以也就无法辅助您的工作。

为什么使用 XML 编辑器?

当今，XML 是非常重要的技术，并且开发项目正在使用这些基于 XML 的技术：

- 用 XML Schema 定义 XML 的结构和数据类型
- 用 XSLT 来转换 XML 数据
- 用 SOAP 来交换应用程序之间的 XML 数据
- 用 WSDL 来描述网络服务
- 用 RDF 来描述网络资源
- 用 XPath 和 XQuery 来访问 XML 数据
- 用 SMIL 来定义图形

为了能够编写出无错的 XML 文档，您需要一款智能的 XML 编辑器！

XML 编辑器

专业的 XML 编辑器会帮助您编写无错的 XML 文档，根据某种 DTD 或者 schema 来验证 XML，以及强制您创建合法的 XML 结构。

XML 编辑器应该能够：

- 为开始标签自动添加结束标签
- 强制您编写合法的 XML
- 根据某种 DTD 来验证 XML
- 根据某种 Schema 来验证 XML
- 对您的 XML 语法进行代码的颜色化



在 W3CSchool，我们多年来一直使用 XMLSpy。 XMLSpy 是我们最喜爱的 XML 编辑器。这里是我们特别喜欢的一些特点：

- 在 32 位和 64 位版本中可用
- 使用方便
- 上下文敏感的人们帮手
- 语法着色和漂亮的印刷
- 智能修复验证与自动校正错误
- 文本视图和网格视图之间轻松切换
- 图形化的 XML Schema 编辑器
- 所有主流数据库的数据库导入导出
- SharePoint? 服务器支持
- 内置许多 XML 文档类型的模板
- 显示 XML 数据的图表创建
- XPath 1.0/2.0 的智能自动完成
- XSLT 1.0/2.0 编辑器、分析器和调试器
- XQuery 编辑器、分析器和调试器
- SOAP 客户端和调试器
- 图像化的 WSDL 1.1/2.0 编辑器
- XBRL 验证 & 分类编辑
- 支持 Office 2007 / OOXML
- Java、C++ 和 C# 的代码生成
- HTML5 和 CSS3 支持

了解更多关于 XMLSpy

XMLSpy 是 Altova MissionKit? 的 XML 软件套件的六个工具之一。 了解更多用于 XML 开发的 Altova MissionKit。

XML - E4X

E4X 向 JavaScript 添加了对 XML 的直接支持。

E4X 实例

```
var employees=
<employees>
<person>
<name>Tove</name>
<age>32</age>
</person>
<person>
<name>Jani</name>
<age>26</age>
</person>
</employees>;

document.write(employees.person.(name == "Tove").age);
```

这个实例仅适用于 **Firefox**！

作为一个 JavaScript 对象的 XML

E4X 是正式的 JavaScript 标准，增加了对 XML 的直接支持。

使用 E4X，您可以用声明 Date 或 Array 对象变量的方式声明 XML 对象变量：

```
**var x = new XML()**

var y = new Date()

var z = new Array()
```

E4X 是一个 ECMAScript（JavaScript）标准

ECMAScript 是 JavaScript 的正式名称。ECMA-262（JavaScript 1.3）是在 1999 年 12 月标准化的。

E4X 是 JavaScript 的扩展，增加了对 XML 的直接支持。ECMA-357（E4X）是在 2004 年 6 月标准化的。

ECMA 组织（成立于 1961 年），是专门用于信息和通信技术（ICT）和消费电子（CE）的标准化。 ECMA 制定的标准为：

- JavaScript

- C# 语言
- 国际字符集
- 光盘
- 磁带
- 数据压缩
- 数据通信
- 等等...

没有使用 **E4X**

下面的实例是一个跨浏览器的实例，实例加载一个现有的 XML 文档（"note.xml"）到 XML 解析器，并显示消息说明：

实例

```
var xmlDoc;
//code for Internet Explorer
if (window.ActiveXObject)
{
xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.load("note.xml");
displaymessage();
}
// code for Mozilla, Firefox, etc.
else (document.implementation && document.implementation.createDocument)
{
xmlDoc= document.implementation.createDocument("","",null);
xmlDoc.load("note.xml");
xmlDoc.onload=displaymessage;
}

function displaymessage()
{
document.write(xmlDoc.getElementsByTagName("body")[0].firstChild.nodeValue);
}
```

使用 **E4X**

下面的实例是上面的实例相同，但是使用了 E4X：

```
var xmlDoc=new XML();
xmlDoc.load("note.xml");
document.write(xmlDoc.body);
```

简单多了，是不是？

浏览器支持

Firefox 是目前唯一对 E4X 的支持比较好的浏览器。

目前还没有支持 E4X 的有 **Opera**、**Chrome** 或 **Safari**。

到目前为止，没有迹象显示在 **Internet Explorer** 中对 E4X 的支持。

E4X 的未来

E4X 没有得到广泛的支持。也许它提供的实用功能太少，尚未被其他的解决方案涉及：

- 对于完整的 XML 处理，您还需要学习 XML DOM 和 XPath
- 对于访问 XMLHttpRequests，JSON 是首选的格式。
- 对于简单的文档处理，JQuery 选择更容易。

DTD 教程

DTD 简介

文档类型定义（DTD）可定义合法的XML文档构建模块。它使用一系列合法的元素来定义文档的结构。

DTD 可被成行地声明于 XML 文档中，也可作为一个外部引用。

内部的 DOCTYPE 声明

假如 DTD 被包含在您的 XML 源文件中，它应当通过下面的语法包装在一个 DOCTYPE 声明中：

```
<!DOCTYPE root-element [element-declarations]>
```

带有 DTD 的 XML 文档实例（请在 IE5 以及更高的版本打开，并选择查看源代码）：

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

在您的浏览器中打开此 XML 文件，并选择"查看源代码"命令。

The DTD above is interpreted like this:

- **!DOCTYPE note** (第二行)定义此文档是 **note** 类型的文档。
- **!ELstrongENT note** (第三行)定义 **note** 元素有四个元素："to、from、heading,、body"
- **!ELstrongENT to** (第四行)定义 **to** 元素为 "#PCDATA" 类型
- **!ELstrongENT from** (第五行)定义 **frome** 元素为 "#PCDATA" 类型
- **!ELstrongENT heading** (第六行)定义 **heading** 元素为 "#PCDATA" 类型
- **!ELstrongENT body** (第七行)定义 **body** 元素为 "#PCDATA" 类型

外部文档声明

假如 DTD 位于 XML 源文件的外部，那么它应通过下面的语法被封装在一个 DOCTYPE 定义中：

```
<!DOCTYPE root-element SYSTEM "filename">
```

这个 XML 文档和上面的 XML 文档相同，但是拥有一个外部的 DTD: （点击打开该文件，并选择"查看源代码"命令。）

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

这是包含 DTD 的 "note.dtd" 文件：

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

为什么使用 **DTD**？

通过 DTD，您的每一个 XML 文件均可携带一个有关其自身格式的描述。

通过 DTD，独立的团体可一致地使用某个标准的 DTD 来交换数据。

而您的应用程序也可使用某个标准的 DTD 来验证从外部接收到的数据。

您还可以使用 DTD 来验证您自身的数据。

DTD - XML 构建模块

构建模块最主要的与元素是 XML 和 HTML文档。

XML 文档构建模块

所有的 XML 文档（以及 HTML 文档）均由以下简单的构建模块构成：

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

元素

元素是 XML 以及 HTML 文档的主要构建模块。

HTML 元素的例子是 "body" 和 "table"。XML 元素的例子是 "note" 和 "message" 。元素可包含文本、其他元素或者是空的。空的 HTML 元素的例子是 "hr"、"br" 以及 "img"。

实例:

```
<body>some text</body>

<message>some text</message>
```

属性可提供有关元素的额外信息。

属性总是被置于某元素的开始标签中。属性总是以名称/值的形式成对出现的。下面的 "img" 元素拥有关于源文件的额外信息：

```
<img src="computer.gif" />
```

元素的名称是 "img"。属性的名称是 "src"。属性的值是 "computer.gif"。由于元素本身为空，它被一个 " /" 关闭。

实体

实体是用来定义普通文本的变量。实体引用是对实体的引用。

大多数同学都了解这个 HTML 实体引用：" "。这个"无折行空格"实体在 HTML 中被用于在某个文档中插入一个额外的空格。

当文档被 XML 解析器解析时，实体就会被展开。

实体应用	字符
<	<
>	>
&	&
"	"
'	'

PCDATA

PCDATA 的意思是被解析的字符数据（parsed character data）。

可把字符数据想象为 XML 元素的开始标签与结束标签之间的文本。

PCDATA 是会被解析器解析的文本。这些文本将被解析器检查实体以及标记。

文本中的标签会被当作标记来处理，而实体会被展开。

不过，被解析的字符数据不应当包含任何 &、< 或者 > 字符；需要使用 &、< 以及 > 实体来分别替换它们。

CDATA

CDATA 的意思是字符数据（character data）。

CDATA 是不会被解析器解析的文本。在这些文本中的标签不会被当作标记来对待，其中的实体也不会被展开。

DTD - 元素

在一个 DTD 中，元素通过元素声明来进行声明。

声明一个元素

在 DTD 中，XML 元素通过元素声明来进行声明。元素声明使用下面的语法：

```
<!ELEMENT element-name category>
或
<!ELEMENT element-name (element-content)>
```

空元素

空元素通过类别关键词EMPTY进行声明：

```
<!ELEMENT element-name EMPTY>

实例:

<!ELEMENT br EMPTY>

XML example:

<br />
```

只有 PCDATA 的元素

只有 PCDATA 的元素通过圆括号中的 #PCDATA 进行声明：

```
<!ELEMENT element-name (#PCDATA)>

实例:

<!ELEMENT from (#PCDATA)>
```

带有任何内容的元素

通过类别关键词 ANY 声明的元素，可包含任何可解析数据的组合：

```
<!ELEMENT element-name ANY>

实例:

<!ELEMENT note ANY>
```

带有子元素（序列）的元素

带有一个或多个子元素的元素通过圆括号中的子元素名进行声明：

```
<!ELEMENT element-name (child1)>
或
<!ELEMENT element-name (child1,child2,...)>

实例:

<!ELEMENT note (to,from,heading,body)>
```

当子元素按照由逗号分隔开的序列进行声明时，这些子元素必须按照相同的顺序出现在文档中。在一个完整的声明中，子元素也必须被声明，同时子元素也可拥有子元素。"note" 元素的完整声明是：

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

声明只出现一次的元素

```
<!ELEMENT element-name (child-name)>

实例:

<!ELEMENT note (message)>
```

上面的例子声明了：message 子元素必须出现一次，并且必须只在 "note" 元素中出现一次。

声明最少出现一次的元素

```
<!ELEMENT element-name (child-name+)>

实例:

<!ELEMENT note (message+)>
```

上面的例子中的加号（+）声明了：message 子元素必须在 "note" 元素内出现至少一次。

声明出现零次或多次的元素

```
<!ELEMENT element-name (child-name*)>

实例:

<!ELEMENT note (message*)>
```

上面的例子中的星号（*）声明了：子元素 message 可在 "note" 元素内出现零次或多次。

声明出现零次或一次的元素

```
<!ELEMENT element-name (child-name?)>

实例:

<!ELEMENT note (message?)>
```

上面的例子中的问号(?)声明了：子元素 message 可在 "note" 元素内出现零次或一次。

声明"非.../既..."类型的内容

```
实例:

<!ELEMENT note (to,from,header,(message|body))>
```

上面的例子声明了："note" 元素必须包含 "to" 元素、"from" 元素、"header" 元素，以及非 "message" 元素既 "body" 元素。

声明混合型的内容

```
实例:

<!ELEMENT note (#PCDATA|to|from|header|message)*>
```

上面的例子声明了："note" 元素可包含出现零次或多次的 PCDATA、"to"、"from"、"header" 或者 "message"。

DTD - 属性

在 DTD 中，属性通过 ATTLIST 声明来进行声明。

声明属性

属性声明使用下列语法：

```
<!ATTLIST element-name attribute-name attribute-type attribute-value>

DTD 实例:

<!ATTLIST payment type CDATA "check">

XML 实例:

<payment type="check" />
```

以下是 属性类型的选项：

类型	描述
CDATA	值为字符数据 (character data)
(*en1*|*en2*|..)	此值是枚举列表中的一个值
ID	值为唯一的 id
IDREF	值为另外一个元素的 id
IDREFS	值为其他 id 的列表
NMTOKEN	值为合法的 XML 名称
NMTOKENS	值为合法的 XML 名称的列表
ENTITY	值是一个实体
ENTITIES	值是一个实体列表
NOTATION	此值是符号的名称
xml:	值是一个预定义的 XML 值

默认属性值可使用下列值：

值	解释
值	属性的默认值
#REQUIRED	属性值是必需的
#IMPLIED	属性不是必需的
#FIXED value	属性值是固定的

默认属性值

```
DTD:
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">

合法的 XML:
<square width="100" />
```

在上面的例子中，"square" 被定义为带有 CDATA 类型的 "width" 属性的空元素。如果宽度没有被设定，其默认值为0 。

#REQUIRED

语法

```
<!ATTLIST element-name attribute-name attribute-type #REQUIRED>
```

实例

```
DTD:
<!ATTLIST person number CDATA #REQUIRED>

合法的 XML:
<person number="5677" />

合法的 XML:
<person />
```

假如您不希望强制作者包含属性，并且您没有默认值选项的话，请使用关键词 #IMPLIED。

#IMPLIED

语法

```
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
```

实例

```
DTD:
<!ATTLIST contact fax CDATA #IMPLIED>

合法的 XML:
<contact fax="555-667788" />

合法的 XML:
<contact />
```

假如您不希望强制作者包含属性，并且您没有默认值选项的话，请使用关键词 #IMPLIED。

#FIXED

语法

```
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">
```

实例

```
DTD:
<!ATTLIST sender company CDATA #FIXED "Microsoft">

合法的 XML:
<sender company="Microsoft" />

非法的 XML:
<sender company="W3Schools" />
```

如果您希望属性拥有固定的值，并不允许作者改变这个值，请使用 #FIXED 关键词。如果作者使用了不同的值，XML 解析器会返回错误。

列举属性值

语法

```
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
```

实例

```
DTD:
<!ATTLIST payment type (check|cash) "cash">

XML 例子:
<payment type="check" />
or
<payment type="cash" />
```

如果您希望属性值为一系列固定的合法值之一，请使用列举属性值。

XML 元素 vs. 属性

在XML中，并有没有规定何时使用属性，以及何时使用子元素。

使用元素 vs. 属性

数据可以存储在子元素或属性。

让我们来看下这些实例:

```
<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

在第一个例子中"sex"是一个属性。在后面一个例子中，"sex"是一个子元素。但是两者都提供了相同的信息。

没有特别规定何时使用属性，以及何时使用子元素。我的经验是在HTML重多使用属性，但在XML中，使用子元素，会感觉更像数据信息。

我喜欢的方式

我喜欢在子元素中存储数据

下面的三个XML文档包含完全相同的信息：

本例中使用"date"属性：

```
<note date="12/11/2002">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

本例中使用"date"元素：

```
<note>
  <date>12/11/2002</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

本例中使用了扩展的"date" 元素: (这是我最喜欢的方式):

```
<note>
  <date>
    <day>12</day>
    <month>11</month>
    <year>2002</year>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

避免使用属性?

你应该避免使用属性?

一些属性具有以下问题:

- 属性不能包含多个值（子元素可以）
- 属性不容易扩展（为以后需求的变化）
- 属性无法描述结构（子元素可以）
- 属性更难以操纵程序代码
- 属性值是不容易测试，针对DTD

如果您使用属性作为数据容器，最终的XML文档将难以阅读和维护。 尝试使用 元素 来描述数据。 to describe data. 只有在提供的数据是不相关信息时我们才建议使用属性。

不要这个样子结束（这不是XML应该使用的）：

```
<note day="12" month="11" year="2002"
to="Tove" from="Jani" heading="Reminder"
body="Don't forget me this weekend!">
</note>
```

一个属性规则的例外

规则总是有另外的

关于属性的规则我有一个例外情况。

有时我指定的 ID 应用了元素。这些 ID 应用可在HTML中的很多相同的情况下可作为 NAME 或者 ID 属性来访问 XML 元素。以下实例展示了这种方式：

```
<messages>
<note id="p501">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

<note id="p502">
  <to>Jani</to>
  <from>Tove</from>
  <heading>Re: Reminder</heading>
  <body>I will not!</body>
</note>
</messages>
```

以上实例的XML文件中，ID是只是一个计数器，或一个唯一的标识符，来识别不同的音符，而不是作为数据的一部分。

在这里我想说的是，元数据（关于数据的数据）应当存储为属性，而数据本身应当存储为元素。

DTD - 实体

实体是用于定义引用普通文本或特殊字符的快捷方式的变量。

- 实体引用是对实体的引用。
- 实体可在内部或外部进行声明。

一个内部实体声明

语法

```
<!ENTITY entity-name "entity-value">
```

实例

```
DTD 实例:

<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3CSchool.cc">

XML 实例：

<author>&writer;&copyright;</author>
```

注意： 一个实体由三部分构成: 一个和号 (&), 一个实体名称, 以及一个分号 (;)。

一个外部实体声明

语法

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

实例

```
DTD 实例:

<!ENTITY writer SYSTEM "http://www.w3cschool.cc/entities.dtd">
<!ENTITY copyright SYSTEM "http://www.w3cschool.cc/entities.dtd">

XML example:

<author>&writer;&copyright;</author>
```

DTD 验证

使用 Internet Explorer 可根据某个 DTD 来验证您的 XML。

通过 XML 解析器进行验证

当您试图打开某个 XML 文档时，XML 解析器有可能会产生错误。通过访问 parseError 对象，就可以取回引起错误的确切代码、文本甚至所在的行。

注意： load() 方法用于文件，而 loadXML() 方法用于字符串。

实例

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
xmlDoc.validateOnParse="true";
xmlDoc.load("note_dtd_error.xml");

document.write("<br />Error Code: ");
document.write(xmlDoc.parseError.errorCode);
document.write("<br />Error Reason: ");
document.write(xmlDoc.parseError.reason);
document.write("<br />Error Line: ");
document.write(xmlDoc.parseError.line);
```

查看xml文件

关闭验证

通过把 XML 解析器的 validateOnParse 设置为 "false"，就可以关闭验证。

实例

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
xmlDoc.validateOnParse="false";
xmlDoc.load("note_dtd_error.xml");

document.write("<br />Error Code: ");
document.write(xmlDoc.parseError.errorCode);
document.write("<br />Error Reason: ");
document.write(xmlDoc.parseError.reason);
document.write("<br />Error Line: ");
document.write(xmlDoc.parseError.line);
```

通用的 XML 验证器

为了帮助您验证 XML 文件，我们创建了此 链接，这样你就可以验证任何 XML 文件了。

parseError 对象

您可以在我们的《XML DOM 教程》中阅读更多有关 parseError 对象的信息。

DTD - 来自网络的实例

电视节目表 DTD

由 David Moisan 创造。拷贝自： http://www.davidmoisan.org/

```
<!DOCTYPE TVSCHEDULE [

<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER,DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>

<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
]>
```

报纸文章 DTD

拷贝自：http://www.vervet.com/

```
<!DOCTYPE NEWSPAPER [

<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>

<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>

<!ENTITY NEWSPAPER "Vervet Logic Times">
<!ENTITY PUBLISHER "Vervet Logic Press">
<!ENTITY COPYRIGHT "Copyright 1998 Vervet Logic Press">

]>
```

产品目录 DTD

拷贝自： http://www.vervet.com/

```
<!DOCTYPE CATALOG [

<!ENTITY AUTHOR "John Doe">
<!ENTITY COMPANY "JD Power Tools, Inc.">
<!ENTITY EMAIL "jd@jd-tools.com">

<!ELEMENT CATALOG (PRODUCT+)>

<!ELEMENT PRODUCT
(SPECIFICATIONS+,OPTIONS?,PRICE+,NOTES?)>
<!ATTLIST PRODUCT
NAME CDATA #IMPLIED
CATEGORY (HandTool|Table|Shop-Professional) "HandTool"
PARTNUM CDATA #IMPLIED
PLANT (Pittsburgh|Milwaukee|Chicago) "Chicago"
INVENTORY (InStock|Backordered|Discontinued) "InStock">

<!ELEMENT SPECIFICATIONS (#PCDATA)>
<!ATTLIST SPECIFICATIONS
WEIGHT CDATA #IMPLIED
POWER CDATA #IMPLIED>

<!ELEMENT OPTIONS (#PCDATA)>
<!ATTLIST OPTIONS
FINISH (Metal|Polished|Matte) "Matte"
ADAPTER (Included|Optional|NotApplicable) "Included"
CASE (HardShell|Soft|NotApplicable) "HardShell">

<!ELEMENT PRICE (#PCDATA)>
<!ATTLIST PRICE
MSRP CDATA #IMPLIED
WHOLESALE CDATA #IMPLIED
STREET CDATA #IMPLIED
SHIPPING CDATA #IMPLIED>

<!ELEMENT NOTES (#PCDATA)>

]>
```

XML DOM

XML DOM 简介

XML DOM 定义了访问和处理 XML 文档的标准。

您应当具备的基础知识

在继续学习之前，您应当对下列知识有基本的了解：

- HTML
- XML
- JavaScript

如果您想要首先学习这些项目，请在我们的首页访问这些教程。

什么是 DOM？

DOM 是 W3C（World Wide Web Consortium）标准。

DOM 定义了访问诸如 XML 和 HTML 文档的标准：

"W3C 文档对象模型（*DOM*，全称 *Document Object Model*）是一个使程序和脚本有能力动态地访问和更新文档的内容、结构以及样式的平台和语言中立的接口。*"*

DOM 被分为 3 个不同的部分/级别：

- 核心 DOM - 用于任何结构化文档的标准模型
- XML DOM - 用于 XML 文档的标准模型
- HTML DOM - 用于 HTML 文档的标准模型

DOM 定义了所有文档元素的对象和属性，以及访问它们的方法（接口）。

什么是 HTML DOM？

HTML DOM 定义了所有 HTML 元素的对象和属性，以及访问它们的方法（接口）。

如果您想要学习 HTML DOM，请在我们的首页访问 HTML DOM 教程。

什么是 XML DOM？

XML DOM 是：

- 用于 XML 的标准对象模型
- 用于 XML 的标准编程接口
- 中立于平台和语言
- W3C 标准

XML DOM 定义了所有 XML 元素的对象和属性，以及访问它们的方法（接口）。

换句话说：**XML DOM** 是用于获取、更改、添加或删除 **XML** 元素的标准。

XML DOM 节点

在 DOM 中，XML 文档中的每个成分都是一个节点。

DOM 节点

根据 DOM，XML 文档中的每个成分都是一个节点。

DOM 是这样规定的：

- 整个文档是一个文档节点
- 每个 XML 元素是一个元素节点
- 包含在 XML 元素中的文本是文本节点
- 每一个 XML 属性是一个属性节点
- 注释是注释节点

DOM 实例

请看下面的 XML 文件（books.xml）：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book category="cooking">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
<book category="children">
<title lang="en">Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
<book category="web">
<title lang="en">XQuery Kick Start</title>
<author>James McGovern</author>
<author>Per Bothner</author>
<author>Kurt Cagle</author>
<author>James Linn</author>
<author>Vaidyanathan Nagarajan</author>
<year>2003</year>
<price>49.99</price>
</book>
<book category="web" cover="paperback">
<title lang="en">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```

在上面的 XML 中，根节点是 <bookstore>。文档中的所有其他节点都被包含在 <bookstore> 中。

根节点 <bookstore> 有四个 <book> 节点。

第一个 <book> 节点有四个节点：<title>、<author>、<year> 和 <price>，其中每个节点都包含一个文本节点，"Everyday Italian"、"Giada De Laurentiis"、"2005" 和 "30.00"。

文本总是存储在文本节点中

在 DOM 处理中一个普遍的错误是，认为元素节点包含文本。

不过，元素节点的文本是存储在文本节点中的。

在这个实例中：**<year>2005</year>**，元素节点 <year>，拥有一个值为 "2005" 的文本节点。

"2005" 不是 <year> 元素的值！

XML DOM 节点树

XML DOM 把 XML 文档视为一棵节点树。

树中的所有节点彼此之间都有关系。

XML DOM 节点树

XML DOM 把 XML 文档视为一种树结构。这种树结构被称为节点树。

可通过这棵树访问所有节点。可以修改或删除它们的内容，也可以创建新的元素。

这颗节点树展示了节点的集合，以及它们之间的联系。这棵树从根节点开始，然后在树的最低层级向文本节点长出枝条：



上面的图片表示 XML 文件 books.xml。

父节点、子节点和同级节点

节点树中的节点彼此之间都有层级关系。

父节点、子节点和同级节点用于描述这种关系。父节点拥有子节点，位于相同层级上的子节点称为同级节点（兄弟或姐妹）。

- 在节点树中，顶端的节点称为根节点
- 根节点之外的每个节点都有一个父节点
- 节点可以有任何数量的子节点
- 叶子是没有子节点的节点
- 同级节点是拥有相同父节点的节点

下面的图片展示出节点树的一个部分，以及节点间的关系：



因为 XML 数据是按照树的形式进行构造的，所以可以在不了解树的确切结构且不了解其中包含的数据类型的情况下，对其进行遍历。

您将在本教程稍后的章节学习更多有关遍历节点树的知识。

第一个子节点 - 最后一个子节点

请看下面的 XML 片段：

```
<bookstore>
<book category="cooking">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
</bookstore>
```

在上面的 XML 中，<title> 元素是 <book> 元素的第一个子节点，而 <price> 元素是 <book> 元素的最后一个子节点。

此外，<book> 元素是 <title>、<author>、<year> 和 <price> 元素的父节点。

XML DOM 解析器

大多数浏览器都内建了供读取和操作 XML 的 XML 解析器。

解析器把 XML 转换为 JavaScript 可存取的对象（XML DOM）。

XML 解析器

XML DOM 包含了遍历 XML 树，访问、插入及删除节点的方法（函数）。

然而，在访问和操作 XML 文档之前，它必须加载到 XML DOM 对象。

XML 解析器读取 XML，并把它转换为 XML DOM 对象，这样才可以使用 JavaScript 访问它。

大多数浏览器有一个内建的 XML 解析器。

加载 XML 文档

下面的 JavaScript 片段加载一个 XML 文档（"books.xml"）：

实例

```
if (window.XMLHttpRequest)
{
xhttp=new XMLHttpRequest();
}
else // IE 5/6
{
xhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xhttp.open("GET","books.xml",false);
xhttp.send();
xmlDoc=xhttp.responseXML;
```

代码解释：

- 创建一个 XMLHTTP 对象
- 打开 XMLHTTP 对象
- 发送一个 XML HTTP 请求到服务器
- 设置响应为 XML DOM 对象

加载 XML 字符串

下面的代码加载并解析一个 XML 字符串：

实例

```
if (window.DOMParser)
{
parser=new DOMParser();
xmlDoc=parser.parseFromString(text,"text/xml");
}
else // Internet Explorer
{
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.loadXML(text);
}
```

注意：Internet Explorer 使用 loadXML() 方法来解析 XML 字符串，而其他浏览器使用 DOMParser 对象。

跨域访问

出于安全原因，现代的浏览器不允许跨域访问。

这意味着，网页以及 XML 文件，它必须位于同一台服务器上尝试加载。

W3CSchool 上的实例中所有打开的 XML 文件都是位于 W3CSchool 域上的。

如果您想要在您的网页上使用上面的实例，您加载的 XML 文件必须位于您自己的服务器上。

XML DOM 加载函数

加载 XML 文档中的代码可以存储在一个函数中。

loadXMLDoc() 函数

为了使前一页中的代码易于维护（检查旧的浏览器），它应该写成一个函数：

```
function loadXMLDoc(dname)
{
if (window.XMLHttpRequest)
{
xhttp=new XMLHttpRequest();
}
else
{
xhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xhttp.open("GET",dname,false);
xhttp.send();
return xhttp.responseXML;
}
```

上面的函数可以存储在 HTML 页面的 <head> 部分，并从页面中的脚本调用。

>  上面描述的函数，用于本教程中所有 **XML** 文档实例！

loadXMLDoc() 的外部 JavaScript

为了使上述代码更容易维护，以确保在所有页面中使用相同的代码，我们把函数存储在一个外部文件中。

文件名为 "loadxmldoc.js"，且在 HTML 页面中的 head 部分被加载。然后，页面中的脚本调用 loadXMLDoc() 函数。

下面的实例使用 loadXMLDoc() 函数加载 books.xml：

实例

```
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

_code goes here.....
_
</script>

</body>
</html>
```

如何从 XML 文件中获得数据，将在下一章中讲解。

loadXMLString() 函数

为了使前一页中的代码易于维护（检查旧的浏览器），它应该写成一个函数：

```
function loadXMLString(txt)
{
if (window.DOMParser)
{
parser=new DOMParser();
xmlDoc=parser.parseFromString(txt,"text/xml");
}
else // Internet Explorer
{
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.loadXML(txt);
}
return xmlDoc;
}
```

上面的函数可以存储在 HTML 页面的 <head> 部分，并从页面中的脚本调用。



> 上面描述的函数，用于本教程中所有 **XML** 字符串实例！

loadXMLString() 的外部 JavaScript

我们已经把 loadXMLString() 函数存储在名为 "loadxmlstring.js" 文件中。

实例

```
<html>
<head>
<script src="loadxmlstring.js"></script>
</head>
<body>
<script>
text="<bookstore>"
text=text+"<book>";
text=text+"<title>Everyday Italian</title>";
text=text+"<author>Giada De Laurentiis</author>";
text=text+"<year>2005</year>";
text=text+"</book>";
text=text+"</bookstore>";

xmlDoc=loadXMLString(text);

_code goes here.....

_</script>
</body>
</html>
```

XML DOM - 属性和方法

属性和方法向 XML DOM 定义了编程接口。

编程接口

DOM 把 XML 模拟为一系列节点对象。可通过 JavaScript 或其他编程语言来访问节点。在本教程中，我们使用 JavaScript。

对 DOM 的编程接口是通过一套标准的属性和方法来定义的。

属性经常按照"某事物是什么"的方式来使用（例如节点名是 "book"）。

方法经常按照"对某事物做什么"的方式来使用（例如删除 "book" 节点）。

XML DOM 属性

一些典型的 DOM 属性：

- x.nodeName - x 的名称
- x.nodeValue - x 的值
- x.parentNode - x 的父节点
- x.childNodes - x 的子节点
- x.attributes - x 的属性节点

注释：在上面的列表中，x 是一个节点对象。

XML DOM 方法

- x.getElementsByTagName(*name*) - 获取带有指定标签名称的所有元素
- x.appendChild(*node*) - 向 x 插入子节点
- x.removeChild(*node*) - 从 x 删除子节点

注释：在上面的列表中，x 是一个节点对象。

实例

从 books.xml 中的 <title> 元素获取文本的 JavaScript 代码：

```
txt=xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue
```

在该语句执行后，txt 保存的值是 "Everyday Italian"。

解释：

- **xmlDoc** - 由解析器创建的 XML DOM 对象
- **getElementsByTagName("title")[0]** - 第一个 <title> 元素
- **childNodes[0]** - <title> 元素的第一个子节点（文本节点）
- **nodeValue** - 节点的值（文本本身）

XML DOM - 访问节点

通过 DOM，您能够访问 XML 文档中的每个节点。

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 函数 loadXMLDoc()，位于外部 JavaScript 中，用于加载 XML 文件。

使用节点列表中的索引号来访问节点 本例使用 getElementsByTagname() 方法来获取 "books.xml" 中的第三个 <title> 元素。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>

xmlDoc=loadXMLDoc("books.xml");
x=xmlDoc.getElementsByTagName("title");
document.write(x[2].childNodes[0].nodeValue);

</script>
</body>
</html>
```

使用 length 属性来遍历节点 本例使用 length 属性来遍历 "books.xml" 中的所有 <title> 元素。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title");
for (i=0;i<x.length;i++)
  {
  document.write(x[i].childNodes[0].nodeValue);
  document.write("<br>");
  }
</script>
</body>
</html>
```

查看元素的节点类型 本例使用 nodeType 属性来获取 "books.xml" 中根元素的节点类型。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

document.write(xmlDoc.documentElement.nodeName);
document.write("<br>");
document.write(xmlDoc.documentElement.nodeType);
</script>
</body>
</html>
```

遍历元素节点 本例使用 nodeType 属性来处理 "books.xml" 中的元素节点。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.documentElement.childNodes;
for (i=0;i<x.length;i++)
{
if (x[i].nodeType==1)
  {//Process only element nodes (type 1)
  document.write(x[i].nodeName);
  document.write("<br>");
  }
}
</script>
</body>
</html>
```

使用节点的关系来遍历元素节点 本例使用 nodeType 属性和 nextSibling 属性来处理 "books.xml" 中的元素节点。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("book")[0].childNodes;
y=xmlDoc.getElementsByTagName("book")[0].firstChild;
for (i=0;i<x.length;i++)
{
if (y.nodeType==1)
  {//Process only element nodes (type 1)
  document.write(y.nodeName + "<br>");
  }
y=y.nextSibling;
}
</script>
</body>
</html>
```

访问节点

您可以通过三种方式来访问节点：

1. 通过使用 getElementsByTagName() 方法。

2. 通过循环（遍历）节点树。

3. 通过利用节点的关系在节点树中导航。

getElementsByTagName() 方法

getElementsByTagName() 返回拥有指定标签名的所有元素。

语法

```
_node_.getElementsByTagName(_"tagname"_);
```

实例

下面的实例返回 x 元素下的所有 <title> 元素：

```
x.getElementsByTagName("title");
```

请注意，上面的实例仅返回 x 节点下的 <title> 元素。如需返回 XML 文档中的所有 <title> 元素，请使用：

```
xmlDoc.getElementsByTagName("title");
```

在这里，xmlDoc 就是文档本身（文档节点）。

DOM 节点列表（Node List）

getElementsByTagName() 方法返回节点列表。节点列表是节点的数组。

下面的代码使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中，然后在变量 x 中存储 <title> 节点的一个列表：

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title");
```

可通过索引号访问 x 中的 <title> 元素。如需访问第三个 <title>，您可以编写：

```
y=x[2];
```

注意：该索引从 0 开始。

在本教程后面的章节中，您将学习更多有关节点列表（Node List）的知识。

DOM 节点列表长度（Node List Length）

length 属性定义节点列表的长度（即节点的数量）。

您可以通过使用 length 属性来遍历节点列表：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title");

for (i=0;i<x.length;i++)
{
document.write(x[i].childNodes[0].nodeValue);
document.write("
");
}
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取所有 <title> 元素节点
3. 输出每个 <title> 元素的文本节点的值

节点类型（Node Types）

XML 文档的 **documentElement** 属性石根节点。

节点的 **nodeName** 属性是节点的名称。

节点的 **nodeType** 属性是节点的类型。

您将在本教程的下一章中学习更多有关节点属性的知识。

尝试一下

遍历节点

下面的代码遍历根节点的子节点，同时也是元素节点：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.documentElement.childNodes;

for (i=0;i<x.length;i++)
{
if (x[i].nodeType==1)
{//Process only element nodes (type 1)
document.write(x[i].nodeName);
document.write("
");
}
}
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取根元素的子节点
3. 检查每个子节点的节点类型。如果节点类型是 "1"，则是元素节点
4. 如果是元素节点，则输出节点的名称

导航节点的关系

下面的代码使用节点关系导航节点树：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("book")[0].childNodes;
y=xmlDoc.getElementsByTagName("book")[0].firstChild;

for (i=0;i<x.length;i++)
{
if (y.nodeType==1)
{//Process only element nodes (type 1)
document.write(y.nodeName + "
");
}
y=y.nextSibling;
}
```

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取第一个 book 元素的子节点
3. 把 "y" 变量设置为第一个 book 元素的第一个子节点
4. 对于每个子节点（第一个子节点从 "y" 开始），检查节点类型，如果节点类型为 "1"，则是元素节点
5. 如果是元素节点，则输出该节点的名称
6. 把 "y" 变量设置为下一个同级节点，并再次运行循环

XML DOM 节点信息

nodeName、nodeValue 和 nodeType 属性包含有关节点的信息。

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 函数 loadXMLDoc()，位于外部 JavaScript 中，用于加载 XML 文件。

获取元素节点的节点名称 本例使用 nodeName 属性来获取 "books.xml" 中根元素的节点名称。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

document.write(xmlDoc.documentElement.nodeName);
</script>
</body>
</html>
```

从文本节点获取文本 本例使用 nodeValue 属性来获取 "books.xml" 中第一个 <title> 元素的文本。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
txt=x.nodeValue;
document.write(txt);
</script>
</body>
</html>
```

更改文本节点中的文本 本例使用 nodeValue 属性来更改 "books.xml" 中第一个 <title> 元素的文本。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.nodeValue="Easy Cooking";

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
txt=x.nodeValue;
document.write(txt);
</script>
</body>
</html>
```

获取元素节点的节点名称和类型 本例使用 nodeName 和 nodeType 属性来获取 "books.xml" 中根元素的节点名称和类型。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

document.write(xmlDoc.documentElement.nodeName);
document.write("<br>");
document.write(xmlDoc.documentElement.nodeType);
</script>
</body>
</html>
```

节点的属性

在 XML DOM 中，每个节点都是一个对象。

对象拥有方法和属性，并可通过 JavaScript 进行访问和操作。

三个重要的节点属性是：

- nodeName
- nodeValue
- nodeType

nodeName 属性

nodeName 属性规定节点的名称。

- nodeName 是只读的
- 元素节点的 nodeName 与标签名相同
- 属性节点的 nodeName 是属性的名称
- 文本节点的 nodeName 永远是 #text
- 文本节点的 nodeName 永远是 #document

尝试一下.

nodeValue 属性

nodeValue 属性规定节点的值。

- 元素节点的 nodeValue 是 undefined
- 文本节点的 nodeValue 是文本本身
- 属性节点的 nodeValue 是属性的值

获取元素的值

下面的代码检索第一个 <title> 元素的文本节点的值：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
txt=x.nodeValue;
```

结果：txt = "Everyday Italian"

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取第一个 <title> 元素节点的文本节点
3. 把 txt 变量设置为文本节点的值

更改元素的值

下面的代码更改第一个 <title> 元素的文本节点的值：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.nodeValue="Easy Cooking";
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取第一个 <title> 元素节点的文本节点
3. 更改文本节点的值为 "Easy Cooking"

nodeType 属性

nodeType 属性规定节点的类型。

nodeType 是只读的。

最重要的节点类型是：

节点类型	NodeType
元素	1
属性	2
文本	3
注释	8
文档	9

尝试一下。

XML DOM 节点列表

节点列表由 getElementsByTagName() 方法和 childNodes 属性返回。

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 函数 loadXMLDoc()，位于外部 JavaScript 中，用于加载 XML 文件。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title");
txt=x[0].childNodes[0].nodeValue;
document.write(txt);
</script>
</body>
</html>
```

从第一个 <title> 元素获取文本 本例使用 getElementsByTagName() 方法从 "books.xml" 中的第一个 <title> 元素获取文本。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName('title');
for (i=0;i<x.length;i++)
{
document.write(x[i].childNodes[0].nodeValue);
document.write("<br>");
}
</script>
</body>
</html>
```

使用 length 属性遍历节点 本例使用节点列表和 length 属性来遍历 "books.xml" 中所有的 <title> 元素。

获取元素的属性 本例使用属性列表从 "books.xml" 中的第一个 <book> 元素获取属性。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("book")[0].attributes;
document.write(x.getNamedItem("category").nodeValue);
document.write("<br>" + x.length);
</script>
</body>
</html>
```

DOM 节点列表（Node List）

当使用诸如 childNodes 或 getElementsByTagName() 的属性或方法是，会返回节点列表对象。

节点列表对象表示节点的列表，与 XML 中的顺序相同。

节点列表中的节点使用从 0 开始的索引号进行访问。

下面的图像表示 "books.xml" 中 <title> 元素的节点列表：



下面的代码片段通过使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中，并返回 "books.xml" 中 title 元素的节点列表：

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title");
```

在上面的语句执行之后，x 是节点列表对象。

下面的代码片段从节点列表（x）中的第一个 <title> 元素返回文本：

实例

```
txt=x[0].childNodes[0].nodeValue;
```

在上面的语句执行之后，txt = "Everyday Italian"。

节点列表长度（Node List Length）

节点列表对象会保持自身的更新。如果删除或添加了元素，列表会自动更新。

节点列表的 length 属性是列表中节点的数量。

下面的代码片段通过使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中，并返回 "books.xml" 中 <title> 元素的数量：

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName('title').length;
```

在上面的语句执行之后，x = 4。

节点列表的长度可用于遍历列表中所有的元素。

下面的代码片段使用 length 属性来遍历 <title> 元素的列表：

实例

```
xmlDoc=loadXMLDoc("books.xml");

//the x variable will hold a node list
x=xmlDoc.getElementsByTagName('title');

for (i=0;i<x.length;i++)
{
document.write(x[i].childNodes[0].nodeValue);
document.write("
");
}
```

输出：

```
Everyday Italian
Harry Potter
XQuery Kick Start
Learning XML
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 设置 x 变量来保存所有 title 元素的节点列表
3. 从所有 <title> 元素的文本节点输出值

DOM 属性列表（命名节点图 Named Node Map）

元素节点的 attributes 属性返回属性节点的列表。

这被称为命名节点图（Named Node Map），除了方法和属性上的一些差别以外，它与节点列表相似。

属性列表会保持自身的更新。如果删除或添加属性，这个列表会自动更新。

下面的代码片段通过使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中，并返回 "books.xml" 中第一个 <book> 元素的属性节点列表：

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName('book')[0].attributes;
```

在上面的代码执行之后，x.length 等于属性的数量，可使用 x.getNamedItem() 返回属性节点。

下面的代码片段显示一个 book 的 "category" 属性的值，以及其属性的数量：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("book")[0].attributes;

document.write(x.getNamedItem("category").nodeValue);
document.write("
" + x.length);
```

输出：

```
cooking
1
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 设置 x 变量来保存第一个 <book> 元素的所有属性的一个列表
3. 从 "category" 属性输出值
4. 输出属性列表的长度

XML DOM 遍历节点树

遍历（Traverse）意味着在节点树中进行循环或移动。

遍历节点树

通常您想要循环 XML 文档，比如：当您需要提取每个元素的值时。

这叫做"遍历节点树"。

下面的实例遍历 <book> 的所有子节点，并显示他们的名称和值：

实例

```
<html>
<head>
<script src="loadxmlstring.js"></script>
</head>
<body>
<script>
text="<book>";
text=text+"<title>Everyday Italian</title>";
text=text+"<author>Giada De Laurentiis</author>";
text=text+"<year>2005</year>";
text=text+"</book>";

xmlDoc=loadXMLString(text);

// documentElement always represents the root node
x=xmlDoc.documentElement.childNodes;
for (i=0;i<x.length;i++)
{
document.write(x[i].nodeName);
document.write(": ");
document.write(x[i].childNodes[0].nodeValue);
document.write("
");
}
</script>
</body>
</html>
```

输出：

```
title: Everyday Italian
author: Giada De Laurentiis
year: 2005
```

实例解释：

1. loadXMLString() 把 XML 字符串载入 xmlDoc 中

2. 获取根元素的子节点
3. 输出每个子节点的节点名称以及文本节点的节点值

XML DOM 浏览器差异

DOM 解析中的浏览器差异

所有现代的浏览器都支持 W3C DOM 规范。

然而，浏览器之间是有差异的。一个重要的差异是：

- 处理空白和换行的方式

DOM - 空白和换行

XML 经常在节点之间包含换行或空白字符。这是在使用简单的编辑器（比如记事本）编辑文档时经常出现的情况。

下面的例子（由记事本编辑）在每行之间包含 CR/LF（换行），在每个子节点之前包含两个空格：

```
<book>
<title>Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
```

Internet Explorer 将不会把空的空白或换行作为文本节点，而其他浏览器会。

下面的代码片段显示（books.xml 的）根元素拥有多少个子节点：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.documentElement.childNodes;
document.write("Number of child nodes: " + x.length);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取根元素的子节点
3. 输出子节点的数量。结果取决于您所使用的浏览器。IE 浏览器会输出 4（提醒 4 个子节点），而其他浏览器会输出 9（提醒 9 个子节点）。

XML DOM - 导航节点

可通过使用节点间的关系对节点进行导航。

导航 DOM 节点

通过节点间的关系访问节点树中的节点，通常称为导航节点（"navigating nodes"）。

在 XML DOM 中，节点的关系被定义为节点的属性：

- parentNode
- childNodes
- firstChild
- lastChild
- nextSibling
- previousSibling

下面的图像展示了 books.xml 中节点树的一个部分，并说明了节点之间的关系：



DOM - 父节点

所有的节点都仅有一个父节点。下面的代码导航到 <book> 的父节点：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("book")[0];
document.write(x.parentNode.nodeName);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取第一个 <book> 元素
3. 输出 "x" 的父节点的节点名称

避免空的文本节点

Firefox 以及其他一些浏览器，把空的空白或换行当作文本节点，而 Internet Explorer 不会这么做。

这会在使用以下属性：firstChild、lastChild、nextSibling、previousSibling 时产生一个问题。

为了避免导航到空的文本节点（元素节点之间的空格和换行符），我们使用一个函数来检查节点类型：

```
function get_nextSibling(n)
{
y=n.nextSibling;
while (y.nodeType!=1)
{
y=y.nextSibling;
}
return y;
}
```

上面的函数允许您使用 get*nextSibling*（_*node*）来代替 *node*.nextSibling 属性。

代码解释：

元素节点的类型是 1。如果同级节点不是元素节点，就移动到下一个节点，直到找到元素节点为止。通过这个办法，在 Internet Explorer 和 Firefox 中，都可以得到相同的结果。

获取第一个子元素

下面的代码显示第一个 <book> 的第一个元素：

实例

```
<html>
<head>
<script src="loadxmldoc.js">
</script>
<script>
//check if the first node is an element node
function get_firstChild(n)
{
y=n.firstChild;
while (y.nodeType!=1)
{
y=y.nextSibling;
}
return y;
}
</script>
</head>

<body>
<script>
xmlDoc=loadXMLDoc("books.xml");

x=get_firstChild(xmlDoc.getElementsByTagName("book")[0]);
document.write(x.nodeName);
</script>
</body>
</html>
```

输出：

```
title
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 在第一个 <book> 元素上使用 get_firstChild 函数，来获取第一个子节点（属于元素节点）
3. 输出第一个子节点（属于元素节点）的节点名称

更多实例

lastChild() 本例使用 lastChild() 方法和一个自定义函数来获取节点的最后一个子节点

nextSibling() 本例使用 nextSibling() 方法和一个自定义函数来获取节点的下一个同级节点

previousSibling() 本例使用 previousSibling() 方法和一个自定义函数来获取节点的上一个同级节点

XML DOM 获取节点值

nodeValue 属性用于获取节点的文本值。

getAttribute() 方法返回属性的值。

获取元素的值

在 DOM 中，每种成分都是节点。元素节点没有文本值。

元素节点的文本存储在子节点中。该节点称为文本节点。

获取元素文本的方法，就是获取这个子节点（文本节点）的值。

获取元素值

getElementsByTagName() 方法返回包含拥有指定标签名的所有元素的节点列表，其中的元素的顺序是它们在源文档中出现的顺序。

下面的代码通过使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中并检索第一个 <title> 元素：

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0];
```

childNodes 属性返回子节点的列表。<title> 元素只有一个子节点。它是一个文本节点。

下面的代码检索 <title> 元素的文本节点：

```
x=xmlDoc.getElementsByTagName("title")[0];
y=x.childNodes[0];
```

nodeValue 属性返回文本节点的文本值：

实例

```
x=xmlDoc.getElementsByTagName("title")[0];
y=x.childNodes[0];
txt=y.nodeValue;
```

结果：txt = "Everyday Italian"

遍历所有 <title> 元素： 尝试一下

获取属性的值

在 DOM 中，属性也是节点。与元素节点不同，属性节点拥有文本值。

获取属性的值的方法，就是获取它的文本值。

可以通过使用 getAttribute() 方法或属性节点的 nodeValue 属性来完成这个任务。

获取属性值 - getAttribute()

getAttribute() 方法返回属性值。

下面的代码检索第一个 <title> 元素的 "lang" 属性的文本值：

实例

```
xmlDoc=loadXMLDoc("books.xml");

txt=xmlDoc.getElementsByTagName("title")[0].getAttribute("lang");
```

结果：txt = "en"

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 把 txt 变量设置为第一个 title 元素节点的 "lang" 属性的值

遍历所有的 <book> 元素，并获取它们的 "category" 属性： 尝试一下

获取属性值 - getAttributeNode()

getAttributeNode() 方法返回属性节点。

下面代码检索第一个 <title> 元素的 "lang" 属性的文本值：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].getAttributeNode("lang");
txt=x.nodeValue;
```

结果：Result: txt = "en"

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取第一个 <title> 元素节点的 "lang" 属性节点
3. 把 txt 变量设置为属性的值

遍历所有的 <book> 元素并获取它们的 "category" 属性： 尝试一下

XML DOM 改变节点值

nodeValue 属性用于改变节点值。

setAttribute() 方法用于改变属性值。

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 函数 loadXMLDoc()，位于外部 JavaScript 中，用于加载 XML 文件。

改变元素的文本节点 本例使用 nodeValue 属性来改变 "books.xml" 中第一个 <title> 元素的文本节点。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.nodeValue="Easy Cooking";

document.write(x.nodeValue);
</script>
</body>
</html>
```

通过使用 setAttribute 来改变属性值 本例使用 setAttribute() 方法来改变第一个 <book> 的 "category" 属性的值。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName('book');

x[0].setAttribute("category","food");

document.write(x[0].getAttribute("category"));
</script>
</body>
</html>
```

通过使用 nodeValue 来改变属性值 本例使用 nodeValue 属性来改变第一个 <book> 的 "category" 属性的值。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("book")[0]
y=x.getAttributeNode("category");
y.nodeValue="food";

document.write(y.nodeValue);
</script>
</body>
</html>
```

改变元素的值

在 DOM 中，每种成分都是节点。元素节点没有文本值。

元素节点的文本存储在子节点中。该节点称为文本节点。

改变元素文本的方法，就是改变这个子节点（文本节点）的值。

改变文本节点的值

nodeValue 属性可用于改变文本节点的值。

下面的代码片段改变了第一个 <title> 元素的文本节点值：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.nodeValue="Easy Cooking";
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取第一个 <title> 元素的文本节点
3. 把该文本节点的节点值更改为 "Easy Cooking"

遍历并更改所有 <title> 元素的文本节点： 尝试一下

改变属性的值

在 DOM 中，属性也是节点。与元素节点不同，属性节点拥有文本值。I

改变属性的值的方法，就是改变它的文本值。

可以通过使用 setAttribute() 方法或属性节点的 nodeValue 属性来完成这个任务。

通过使用 **setAttribute()** 改变属性

setAttribute() 方法改变已有属性的值，或创建新属性。

下面的代码改变 <book> 元素的 category 属性：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName('book');
x[0].setAttribute("category","food");
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取第一个 <book> 元素
3. 把 "category" 属性的值更改为 "food"

遍历所有的 <title> 元素并添加一个新属性： 尝试一下

注意：如果属性不存在，则创建一个新属性（拥有指定的名称和值）。

通过使用 nodeValue 改变属性

nodeValue 属性可用于更改属性节点的值：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("book")[0]
y=x.getAttributeNode("category");
y.nodeValue="food";
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取第一个 <book> 元素的 "category" 属性
3. 把该属性节点的值更改为 "food"

XML DOM 删除节点

removeChild() 方法删除指定节点。

removeAttribute() 方法删除指定属性。

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 函数 loadXMLDoc()，位于外部 JavaScript 中，用于加载 XML 文件。

删除元素节点 本例使用 removeChild() 来删除第一个 <book> 元素。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>
<script>
xmlDoc=loadXMLDoc("books.xml");

document.write("Number of book nodes: ");
document.write(xmlDoc.getElementsByTagName('book').length);
document.write("<br>");

y=xmlDoc.getElementsByTagName("book")[0];
xmlDoc.documentElement.removeChild(y);

document.write("Number of book nodes after removeChild(): ");
document.write(xmlDoc.getElementsByTagName('book').length);

</script>
</body>
</html>
```

删除当前元素节点 本例使用 parentNode 和 removeChild() 来删除当前的 <book> 元素。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>
<script>
xmlDoc=loadXMLDoc("books.xml");

document.write("Number of book nodes before removeChild(): ");
document.write(xmlDoc.getElementsByTagName("book").length);
document.write("<br>");

x=xmlDoc.getElementsByTagName("book")[0]
x.parentNode.removeChild(x);

document.write("Number of book nodes after removeChild(): ");
document.write(xmlDoc.getElementsByTagName("book").length);

</script>
</body>
</html>
```

删除文本节点 本例使用 removeChild() 来删除第一个 <title> 元素的文本节点。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>
<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0];

document.write("Child nodes: ");
document.write(x.childNodes.length);
document.write("<br>");

y=x.childNodes[0];
x.removeChild(y);

document.write("Child nodes: ");
document.write(x.childNodes.length);
</script>
</body>
</html>
```

清空文本节点的文本 本例使用 nodeValue() 属性来清空第一个 <title> 元素的文本节点。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];

document.write("Value: " + x.nodeValue);
document.write("<br>");

x.nodeValue="";

document.write("Value: " + x.nodeValue);
</script>
</body>
</html>
```

根据名称删除属性 本例使用 removeAttribute() 从第一个 <book> 元素中删除 "category" 属性。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName('book');

document.write(x[0].getAttribute('category'));
document.write("<br>");

x[0].removeAttribute('category');

document.write(x[0].getAttribute('category'));

</script>
</body>
</html>
```

根据对象删除属性 本例使用 removeAttributeNode() 从所有 <book> 元素中删除所有属性。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName('book');

for (i=0;i<x.length;i++)
{
while (x[i].attributes.length>0)
  {
  attnode=x[i].attributes[0];
  old_att=x[i].removeAttributeNode(attnode);

  document.write("Removed: " + old_att.nodeName)
  document.write(": " + old_att.nodeValue)
  document.write("<br>")
  }
}
</script>
</body>
</html>
```

删除元素节点

removeChild() 方法删除指定的节点。

当一个节点被删除时，其所有子节点也会被删除。

下面的代码片段将从载入的 xml 中删除第一个 <book> 元素：

实例

```
xmlDoc=loadXMLDoc("books.xml");

y=xmlDoc.getElementsByTagName("book")[0];

xmlDoc.documentElement.removeChild(y);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 把变量 y 设置为要删除的元素节点
3. 通过使用 removeChild() 方法从父节点删除元素节点

删除自身 - 删除当前的节点

removeChild() 方法是唯一可以删除指定节点的方法。

当您已导航到需要删除的节点时，就可以通过使用 parentNode 属性和 removeChild() 方法来删除此节点：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("book")[0];

x.parentNode.removeChild(x);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 把变量 y 设置为要删除的元素节点
3. 通过使用 parentNode 属性和 removeChild() 方法来删除此元素节点

删除文本节点

removeChild() 方法可用于删除文本节点：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0];

y=x.childNodes[0];
x.removeChild(y);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 把变量 x 设置为第一个 title 元素节点
3. 把变量 y 设置为要删除的文本节点
4. 通过使用 removeChild() 方法从父节点删除元素节点

不太常用 removeChild() 从节点删除文本。可以使用 nodeValue 属性代替它。请看下一段。

清空文本节点

nodeValue 属性可用于改变或清空文本节点的值：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.nodeValue="";
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 把变量 x 设置为第一个 title 元素的文本节点
3. 使用 nodeValue 属性来清空文本节点的文本

遍历并更改所有 <title> 元素的文本节点： 尝试一下

根据名称删除属性节点

removeAttribute(*name*) 方法用于根据名称删除属性节点。

实例：removeAttribute('category')

下面的代码片段删除第一个 <book> 元素中的 "category" 属性：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("book");
x[0].removeAttribute("category");
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 使用 getElementsByTagName() 来获取 book 节点
3. 从第一个 book 元素节点中删除 "category" 属性

遍历并删除所有 <book> 元素的 "category" 属性： 尝试一下

根据对象删除属性节点

removeAttributeNode(*node*) 方法通过使用 node 对象作为参数，来删除属性节点。

实例: removeAttributeNode(x)

下面的代码片段删除所有 <book> 元素的所有属性：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("book");

for (i=0;i<x.length;i++)
{
while (x[i].attributes.length>0)
{
attnode=x[i].attributes[0];
old_att=x[i].removeAttributeNode(attnode);
}
}
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 使用 getElementsByTagName() 来获取所有 book 节点
3. 检查每个 book 元素是否拥有属性
4. 如果在某个 book 元素中存在属性，则删除该属性

XML DOM 替换节点

replaceChild() 方法替换指定节点。

nodeValue 属性替换文本节点中的文本。

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 函数 loadXMLDoc()，位于外部 JavaScript 中，用于加载 XML 文件。

替换元素节点 本例使用 replaceChild() 来替换第一个 <book> 节点。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.documentElement;

//create a book element, title element and a text node
newNode=xmlDoc.createElement("book");
newTitle=xmlDoc.createElement("title");
newText=xmlDoc.createTextNode("A Notebook");

//add the text node to the title node,
newTitle.appendChild(newText);
//add the title node to the book node
newNode.appendChild(newTitle);

y=xmlDoc.getElementsByTagName("book")[0]
//replace the first book node with the new node
x.replaceChild(newNode,y);

z=xmlDoc.getElementsByTagName("title");
for (i=0;i<z.length;i++)
{
document.write(z[i].childNodes[0].nodeValue);
document.write("<br>");
}
</script>
</body>
</html>
```

替换文本节点中的数据 本例使用 nodeValue 属性来替换文本节点中的数据。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
document.write(x.nodeValue);
x.nodeValue="Easy Italian";

document.write("<br>");
document.write(x.nodeValue);

</script>
</body>
</html>
```

替换元素节点

replaceChild() 方法用于替换节点。

下面的代码片段替换第一个 <book> 元素：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.documentElement;

//create a book element, title element and a text node
newNode=xmlDoc.createElement("book");
newTitle=xmlDoc.createElement("title");
newText=xmlDoc.createTextNode("A Notebook");

//add the text node to the title node,
newTitle.appendChild(newText);
//add the title node to the book node
newNode.appendChild(newTitle);

y=xmlDoc.getElementsByTagName("book")[0]
//replace the first book node with the new node
x.replaceChild(newNode,y);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 创建一个新的元素节点 <book>
3. 创建一个新的元素节点 <title>
4. 创建一个新的文本节点，带有文本 "A Notebook"
5. 向新元素节点 <title> 追加这个新文本节点

6. 向新元素节点 <book> 追加这个新元素节点 <title>
7. 把第一个 <book> 元素节点替换为新的 <book> 元素节点

替换文本节点中的数据

replaceData() 方法用于替换文本节点中的数据。

replaceData() 方法有三个参数：

- offset - 在何处开始替换字符。offset 值以 0 开始。
- length - 要替换多少字符
- string - 要插入的字符串

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];

x.replaceData(0,8,"Easy");
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取第一个 <title> 元素节点的文本节点
3. 使用 replaceData 方法把文本节点的前 8 个字符替换为 "Easy"

使用 nodeValue 属性代替

用 nodeValue 属性来替换文本节点中数据会更加容易。

下面的代码片段将用 "Easy Italian" 替换第一个 <title> 元素中的文本节点值：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];

x.nodeValue="Easy Italian";
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中

2. 获取第一个 <title> 元素节点的文本节点
3. 使用 nodeValue 属性来更改这个文本节点的文本

您可以在改变节点这一章中阅读更多有关更改节点值的内容。

XML DOM 创建节点

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 函数 loadXMLDoc()，位于外部 JavaScript 中，用于加载 XML 文件。

创建元素节点 本例使用 createElement() 来创建一个新的元素节点，并使用 appendChild() 把它添加到一个节点中。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

newel=xmlDoc.createElement("edition");
newtext=xmlDoc.createTextNode("first");
newel.appendChild(newtext);

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newel);

//Output title and edition
document.write(x.getElementsByTagName("title")[0].childNodes[0].nodeValue);
document.write(" - Edition: ");
document.write(x.getElementsByTagName("edition")[0].childNodes[0].nodeValue);
</script>
</body>
</html>
```

使用 createAttribute 创建属性节点 本例使用 createAttribute() 来创建一个新的属性节点，并使用 setAttributeNode() 把它插入一个元素中。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

newatt=xmlDoc.createAttribute("edition");
newatt.nodeValue="first";

x=xmlDoc.getElementsByTagName("title");
x[0].setAttributeNode(newatt);

document.write("Edition: ");
document.write(x[0].getAttribute("edition"));

</script>
</body>
</html>
```

使用 setAttribute 创建属性节点 本例使用 setAttribute() 为一个元素创建一个新的属性。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title");

x[0].setAttribute("edition","first");

document.write("Edition: ");
document.write(x[0].getAttribute("edition"));

</script>
</body>
</html>
```

创建文本节点 本例使用 createTextNode() 来创建一个新的文本节点，并使用 appendChild() 把它添加到一个元素中。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

newel=xmlDoc.createElement("edition");
newtext=xmlDoc.createTextNode("first");
newel.appendChild(newtext);

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newel);

//Output title and edition
document.write(x.getElementsByTagName("title")[0].childNodes[0].nodeValue);
document.write(" - Edition: ");
document.write(x.getElementsByTagName("edition")[0].childNodes[0].nodeValue);
</script>
</body>
</html>
```

创建 CDATA section 节点 本例使用 createCDATAsection() 来创建一个 CDATA section 节点，并使用 appendChild() 把它添加到一个元素中。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

newCDATA=xmlDoc.createCDATASection("Special Offer & Book Sale");

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newCDATA);

document.write(x.lastChild.nodeValue);
</script>
</body>
</html>
```

创建注释节点 本例使用 createComment() 来创建一个注释节点，并使用 appendChild() 把它添加到一个元素中。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

newComment=xmlDoc.createComment("Revised April 2008");

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newComment);

document.write(x.lastChild.nodeValue);
</script>
</body>
</html>
```

创建新的元素节点

createElement() 方法创建一个新的元素节点：

实例

```
xmlDoc=loadXMLDoc("books.xml");

newel=xmlDoc.createElement("edition");

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newel);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 创建一个新的元素节点 <edition>
3. 向第一个 <book> 元素追加这个元素节点

遍历并向所有 <book> 元素添加一个元素： 尝试一下

创建新的属性节点

createAttribute() 用于创建一个新的属性节点：

实例

```
xmlDoc=loadXMLDoc("books.xml");

newatt=xmlDoc.createAttribute("edition");
newatt.nodeValue="first";

x=xmlDoc.getElementsByTagName("title");
x[0].setAttributeNode(newatt);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 创建一个新的属性节点 "edition"
3. 设置属性节点的值为 "first"
4. 向第一个 <title> 元素添加这个新的属性节点

遍历所有的 <title> 元素，并添加一个新的属性节点： 尝试一下

注意：如果该属性已存在，则被新属性替代。

使用 **setAttribute()** 创建属性

由于 setAttribute() 方法可以在属性不存在的情况下创建新的属性，我们可以使用这个方法来创建一个新的属性。

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName('book');
x[0].setAttribute("edition","first");
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 为第一个 <book> 元素设置（创建）值为 "first" 的 "edition" 属性

遍历所有的 <title> 元素并添加一个新属性： 尝试一下

创建文本节点

createTextNode() 方法创建一个新的文本节点：

实例

```
xmlDoc=loadXMLDoc("books.xml");

newel=xmlDoc.createElement("edition");
newtext=xmlDoc.createTextNode("first");
newel.appendChild(newtext);

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newel);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 创建一个新元素节点 <edition>
3. 创建一个新的文本节点，其文本是 "first"
4. 向这个元素节点追加新的文本节点
5. 向第一个 <book> 元素追加新的元素节点

向所有的 <book> 元素添加一个带有文本节点的元素节点： 尝试一下

创建 CDATA Section 节点

createCDATASection() 方法创建一个新的 CDATA section 节点。

实例

```
xmlDoc=loadXMLDoc("books.xml");

newCDATA=xmlDoc.createCDATASection("Special Offer & Book Sale");

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newCDATA);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 创建一个新的 CDATA section 节点
3. 向第一个 <book> 元素追加这个新的 CDATA section 节点

遍历并向所有 <book> 元素添加一个 CDATA section： 尝试一下

创建注释节点

createComment() 方法创建一个新的注释节点。

实例

```
xmlDoc=loadXMLDoc("books.xml");

newComment=xmlDoc.createComment("Revised March 2008");

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newComment);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 创建一个新的注释节点
3. 把这个新的注释节点追加到第一个 <book> 元素

循环并向所有 <book> 元素添加一个注释节点： 尝试一下

XML DOM 添加节点

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 函数 loadXMLDoc()，位于外部 JavaScript 中，用于加载 XML 文件。

在最后一个子节点之后添加一个节点 本例使用 appendChild() 方法向一个已有的节点添加一个子节点。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

newel=xmlDoc.createElement("edition");

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newel);

document.write(x.getElementsByTagName("edition")[0].nodeName);
</script>
</body>
</html>
```

在指定的子节点之前添加一个节点 本例使用 insertBefore() 方法在一个指定的子节点之前插入一个节点。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

newNode=xmlDoc.createElement("book");

x=xmlDoc.documentElement;
y=xmlDoc.getElementsByTagName("book");

document.write("Book elements before: " + y.length);
document.write("<br>");
x.insertBefore(newNode,y[3]);

y=xmlDoc.getElementsByTagName("book");
document.write("Book elements after: " + y.length);
</script>
</body>
</html>
```

添加一个新属性 本例使用 setAttribute() 方法添加一个新的属性。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title");

x[0].setAttribute("edition","first");

document.write("Edition: ");
document.write(x[0].getAttribute("edition"));

</script>
</body>
</html>
```

向文本节点添加数据 本例使用 insertData() 把数据插入一个已有的文本节点中。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
document.write(x.nodeValue);
x.insertData(0,"Easy ");
document.write("<br>");
document.write(x.nodeValue);

</script>
</body>
</html>
```

添加节点 - appendChild()

appendChild() 方法向一个已有的节点添加一个子节点。

新节点会添加（追加）到任何已有的子节点之后。

注意：如果节点的位置很重要，请使用 insertBefore() 方法。

下面的代码片段创建一个元素（<edition>），并把它添加到第一个 <book> 元素的最后一个子节点后面：

实例

```
xmlDoc=loadXMLDoc("books.xml");

newel=xmlDoc.createElement("edition");

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newel);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 创建一个新节点 <edition>
3. 把这个节点追加到第一个 <book> 元素

遍历并向所有 <book> 元素追加一个元素：尝试一下

插入节点 - insertBefore()

insertBefore()方法用于在指定的子节点之前插入节点。

在被添加的节点的位置很重要时，此方法很有用：

实例

```
xmlDoc=loadXMLDoc("books.xml");

newNode=xmlDoc.createElement("book");

x=xmlDoc.documentElement;
y=xmlDoc.getElementsByTagName("book")[3];

x.insertBefore(newNode,y);
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 创建一个新的元素节点 <book>
3. 把这个新节点插到最后一个 <book> 元素节点之前

如果 insertBefore() 的第二个参数是 null，新节点将被添加到最后一个已有的子节点之后。

x.insertBefore(newNode,null) 和 **x.appendChild(newNode)** 都可以向 x 追加一个新的子节点。

添加新属性

addAtribute() 这个方法是不存在的。

如果属性不存在，则 setAttribute() 可创建一个新的属性：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName('book');
x[0].setAttribute("edition","first");
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 把第一个 <book> 元素的 "edition" 属性的值设置（创建）为 "first"

注意：如果属性已存在，setAttribute() 方法将覆盖已有的值。

向文本节点添加文本 - insertData()

insertData() 方法将数据插入已有的文本节点中。

insertData() 方法有两个参数：

- offset - 在何处开始插入字符（以 0 开始）
- string - 要插入的字符串

下面的代码片段将把 "Easy" 添加到已加载的 XML 的第一个 <title> 元素的文本节点：

实例

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];

x.insertData(0,"Easy ");
```

XML DOM 克隆节点

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 函数 loadXMLDoc()，位于外部 JavaScript 中，用于加载 XML 文件。

复制一个节点，并把它追加到已有的节点 本例使用 cloneNode() 来复制一个节点，并把它追加到 XML 文档的根节点。

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");
x=xmlDoc.getElementsByTagName('book')[0];
cloneNode=x.cloneNode(true);
xmlDoc.documentElement.appendChild(cloneNode);

//Output all titles
y=xmlDoc.getElementsByTagName("title");
for (i=0;i<y.length;i++)
{
document.write(y[i].childNodes[0].nodeValue);
document.write("<br>");
}
</script>
</body>
</html>
```

复制节点

cloneNode() 方法创建指定节点的副本。

cloneNode() 方法有一个参数（true 或 false）。该参数指示被克隆的节点是否包括原节点的所有属性和子节点。

下面的代码片段复制第一个 <book> 节点，并把它追加到文档的根节点：

实例

```
xmlDoc=loadXMLDoc("books.xml");

oldNode=xmlDoc.getElementsByTagName('book')[0];
newNode=oldNode.cloneNode(true);
xmlDoc.documentElement.appendChild(newNode);

//Output all titles
y=xmlDoc.getElementsByTagName("title");
for (i=0;i<y.length;i++)
{
document.write(y[i].childNodes[0].nodeValue);
document.write("
");
}
```

输出：

```
Everyday Italian
Harry Potter
XQuery Kick Start
Learning XML
Everyday Italian
```

实例解释：

1. 使用 loadXMLDoc() 把 "books.xml" 载入 xmlDoc 中
2. 获取要复制的节点
3. 使用 cloneNode 方法把节点复制到 "newNode" 中
4. 向 XML 文档的根节点追加新的节点
5. 输出文档中所有 book 的所有 title

The XMLHttpRequest 对象

通过 XMLHttpRequest 对象，您可以在不重新加载整个页面的情况下更新网页中的某个部分。

XMLHttpRequest 对象

XMLHttpRequest 对象用于幕后与服务器交换数据。

XMLHttpRequest 对象是开发者的梦想，因为您可以：

- 在不重新加载页面的情况下更新网页
- 在页面已加载后从服务器请求数据
- 在页面已加载后从服务器接收数据
- 在后台向服务器发送数据

创建 XMLHttpRequest 对象

所有现代的浏览器（IE7+、Firefox、Chrome、Safari 和 Opera）都有一个内建的 XMLHttpRequest 对象。

创建 XMLHttpRequest 对象的语法

```
xmlhttp=new XMLHttpRequest();
```

旧版本的 Internet Explorer（IE5 和 IE6）使用 ActiveX 对象：

```
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
```

为了处理所有现代的浏览器，包括 IE5 和 IE6，请检查浏览器是否支持 XMLHttpRequest 对象。如果支持，则创建一个 XMLHttpRequest 对象，如果不支持，则创建一个 ActiveX 对象：

实例

```
if (window.XMLHttpRequest)
{// code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp=new XMLHttpRequest();
}
else
{// code for IE6, IE5
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
```

发送一个请求到服务器

为了发送一个请求到服务器，我们使用 XMLHttpRequest 对象的 open() 和 send() 方法：

```
xmlhttp.open("GET","xmlhttp_info.txt",true);
xmlhttp.send();
```

方法	描述
open(*method*,*url*,*async*)	规定请求的类型，URL，请求是否应该进行异步处理。

method：请求的类型：GET 或 POST *url*：文件在服务器上的位置 *async*：true（异步）或 false（同步） | | send(*string*) | 发送请求到服务器。

string：仅用于 POST 请求 |

GET 或 POST？

GET 比 POST 简单并且快速，可用于大多数情况下。

然而，下面的情况下请始终使用 POST 请求：

- 缓存的文件不是一个选项（更新服务器上的文件或数据库）
- 发送到服务器的数据量较大（POST 没有大小的限制）
- 发送用户输入（可以包含未知字符），POST 比 GET 更强大更安全

URL - 服务器上的文件

open() 方法的 url 参数，是一个在服务器上的文件的地址：

```
xmlhttp.open("GET","xmlhttp_info.txt",true);
```

该文件可以是任何类型的文件（如 .txt 和 .xml），或服务器脚本文件（如.html 和 .php，可在发送回响应之前在服务器上执行动作）。

异步 - True 或 False?

如需异步发送请求，open() 方法的 async 参数必需设置为 true：

```
xmlhttp.open("GET","xmlhttp_info.txt",true);
```

发送异步请求对于 Web 开发人员是一个巨大的进步。在服务器上执行的许多任务非常费时。

通过异步发送，JavaScript 不需要等待服务器的响应，但可以替换为：

- 等待服务器的响应时，执行其他脚本
- 响应准备时处理响应

Async=true

当使用 async=true 时，在 onreadystatechange 事件中响应准备时规定一个要执行的函数：

实例

```
xmlhttp.onreadystatechange=function()
{
if (xmlhttp.readyState==4 && xmlhttp.status==200)
{
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
}
}
xmlhttp.open("GET","xmlhttp_info.txt",true);
xmlhttp.send();
```

Async=false

如需使用 async=false，请更改 open() 方法的第三个参数为 false：

```
xmlhttp.open("GET","xmlhttp_info.txt",false);
```

不推荐使用 async=false，但如果处理几个小的请求还是可以的。

请记住，JavaScript 在服务器响应准备之前不会继续执行。如果服务器正忙或缓慢，应用程序将挂起或停止。

注意：当您使用 async=false 时，不要编写 onreadystatechange 函数 - 只需要把代码放置在 send() 语句之后即可：

实例

```
xmlhttp.open("GET","xmlhttp_info.txt",false);
xmlhttp.send();
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

服务器响应

如需从服务器获取响应，请使用 XMLHttpRequest 对象的 responseText 或 responseXML 属性。

属性	描述
responseText	获取响应数据作为字符串
responseXML	获取响应数据作为 XML 数据

responseText 属性

如果来自服务器的响应不是 XML，请使用 responseText 属性。

responseText 属性以字符串形式返回响应，您可以相应地使用它：

实例

```
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

responseXML 属性

如果来自服务器的响应不是 XML，且您想要把它解析为 XML 对象，请使用 responseXML 属性：

实例

请求文件 cd_catalog.xml 并解析响应：

```
xmlDoc=xmlhttp.responseXML;
var txt="";
x=xmlDoc.getElementsByTagName("ARTIST");
for (i=0;i<x.length;i++)
{
txt=txt + x[i].childNodes[0].nodeValue + "
";
}
document.getElementById("myDiv").innerHTML=txt;
```

onreadystatechange 事件

当请求被发送到服务器，我们要根据响应执行某些动作。

onreadystatechange 事件在每次 readyState 变化时被触发。

readyState 属性持有 XMLHttpRequest 的状态。

XMLHttpRequest 对象的三个重要的属性：

属性	描述
onreadystatechange	存储函数（或函数的名称）在每次 readyState 属性变化时被自动调用
readyState	存放了 XMLHttpRequest 的状态。从 0 到 4 变化：

0：请求未初始化 1：服务器建立连接 2：收到的请求 3：处理请求 4：请求完成和响应准备就绪 | | status | 200："OK" 404：找不到页面 |

在 onreadystatechange 事件中，我们规定当服务器的响应准备处理时会发生什么。

当 readyState 是 4 或状态是 200 时，响应准备：

实例

```
xmlhttp.onreadystatechange=function()
{
if (xmlhttp.readyState==4 && xmlhttp.status==200)
{
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
}
}
```

注意：onreadystatechange 事件在每次 readyState 发生变化时被触发，总共触发了四次。

更多实例

通过 getAllResponseHeaders() 检索头信息 检索资源（文件）的头信息。

通过 getResponseHeader() 检索指定头信息 检索资源（文件）的指定头信息。

检索 ASP 文件的内容 当用户在输入字段键入字符时，网页如何与 Web 服务器进行通信。

从数据库中检索内容 网页如何通过 XMLHttpRequest 对象从数据库中提取信息。

检索 XML 文件的内容 创建一个 XMLHttpRequest 从 XML 文件中检索数据并把数据显示在一个 HTML 表格中。

XML DOM 节点类型

DOM 是一个代表节点对象层次的文档。

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 函数 loadXMLDoc()，位于外部 JavaScript 中，用于加载 XML 文件。

显示所有元素的 nodeName 和 nodeType

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

document.write("Nodename: " + xmlDoc.nodeName);
document.write(" (nodetype: " + xmlDoc.nodeType + ")<br>");

x=xmlDoc.documentElement;

document.write("Nodename: " + x.nodeName);
document.write(" (nodetype: " + x.nodeType + ")<br>");

y=x.childNodes;

for (i=0;i<y.length;i++)
{
document.write("Nodename: " + y[i].nodeName);
document.write(" (nodetype: " + y[i].nodeType + ")<br>");
for (z=0;z<y[i].childNodes.length;z++)
  {
  document.write("Nodename: " + y[i].childNodes[z].nodeName);
  document.write(" (nodetype: " + y[i].childNodes[z].nodeType + ")<br>");
  }
}
</script>
</body>
</html>
```

显示所有元素的 nodeName 和 nodeValue

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

document.write("Nodename: " + xmlDoc.nodeName);
document.write(" (value: " + xmlDoc.childNodes[0].nodeValue + ")<br>");

x=xmlDoc.documentElement;

document.write("Nodename: " + x.nodeName);
document.write(" (value: " + x.childNodes[0].nodeValue + ")<br>");

y=xmlDoc.documentElement.childNodes;

for (i=0;i<y.length;i++)
{
if (y[i].nodeType!=3)
  {
  document.write("Nodename: " + y[i].nodeName);
  document.write(" (value: " + y[i].childNodes[0].nodeValue + ")<br>");
  for (z=0;z<y[i].childNodes.length;z++)
    {
    if (y[i].childNodes[z].nodeType!=3)
      {
      document.write("Nodename: " + y[i].childNodes[z].nodeName);
      document.write(" (value: " + y[i].childNodes[z].childNodes[0].nodeValue + ")<br>");
      }
    }
  }
}
</script>
</body>
</html>
```

节点类型

下面的表格列举了不同的 W3C 节点类型，每个节点类型中可能会包含子类：

节点类型	描述	子类
Document	代表整个文档（DOM 树的根节点）	Element (max. one), ProcessingInstruction, Comment, DocumentType
DocumentFragment	代表"轻量级"的 Document 对象，它可以保留文档中的一部分	Element, ProcessingInstruction, Comment, Text, CDATASection, Entity参考手册
DocumentType	为文档中定义的实体提供了一个接口	None
ProcessingInstruction	代表一个处理指令	None
EntityReference	代表一个实体引用	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Element	表示一个元素	Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
Attr	代表一个属性	Text, EntityReference
Text	代表元素或属性的文本内容	None
CDATASection	代表文档中的 CDATA 区段（文本不会被解析器解析）	None
Comment	代表一个注释	None
Entity	代表一个实体	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Notation	定义一个在 DTD 中声明的符号	None

节点类型 - 返回值

下面的表格列举了每个节点类型（nodetype）所返回的节点名称（nodeName）和节点值（nodeValue）：

节点类型	返回的节点名称	返回的节点值
Document	#document	null
DocumentFragment	#document fragment	null
DocumentType	文档类型名称	null
Entity参考手册	实体引用名称	null
Element	元素名称	null
Attr	属性名称	属性值
ProcessingInstruction	目标	节点的内容
Comment	#comment	注释文本
Text	#text	节点的内容
CDATASection	#cdata-section	节点的内容
Entity	实体名称	null
Notation	符号名称	null

节点类型 - 命名常量

节点类型	命名常量
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	ENTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE
12	NOTATION_NODE

XML DOM - Node 对象

Node 对象

Node 对象代表文档树中的一个单独的节点。

这里的节点可以是：元素节点、属性节点、文本节点以及所有在 节点类型这章中所提到的所有其他的节点类型。

请注意，尽管所有的对象都继承了用以处理父节点和子节点的 Node 属性 / 方法，但是并不是所有的对象都可以包含父节点或子节点。举个例子来说，Text 节点中可能不包含子节点，所以把子节点添加到文本节点中可能会导致一个 DOM 错误。

Node 对象属性

属性	描述
baseURI	返回节点的绝对基准 URI。
childNodes	返回节点的子节点的节点列表。
firstChild	返回节点的第一个子节点。
lastChild	返回节点的最后一个子节点。
localName	返回节点名称的本地部分。
namespaceURI	返回节点的命名空间 URI。
nextSibling	返回元素之后紧接的节点。
nodeName	返回节点的名称，根据其类型。
nodeType	返回节点的类型。
nodeValue	设置或返回节点的值，根据其类型。
ownerDocument	返回节点的根元素（document 对象）。
parentNode	返回节点的父节点。
prefix	设置或返回节点的命名空间前缀。
previousSibling	返回元素之前紧接的节点。
textContent	设置或返回节点及其后代的文本内容。

Node 对象方法

方法	描述
appendChild()	把新的子节点添加到节点的子节点列表末尾。
cloneNode()	克隆节点。
compareDocumentPosition()	比较两个节点的文档位置。
getFeature(feature,version)	返回 DOM 对象，此对象可执行带有指定特性和版本的专门的 API。
getUserData(key)	返回与节点上键关联的对象。此对象必须首先通过使用相同的键调用 setUserData 来设置到此节点。
hasAttributes()	如果节点拥有属性，则返回 ture，否则返回 false。
hasChildNodes()	如果节点拥有子节点，则返回 true，否则返回 false。
insertBefore()	在已有的子节点之前插入一个新的子节点。
isDefaultNamespace(URI)	返回指定的 namespaceURI 是否默认。
isEqualNode()	检查两个节点是否相等。
isSameNode()	检查两个节点是否为同一节点。
isSupported(feature,version)	返回指定的特性是否在此节点上得到支持。
lookupNamespaceURI()	返回匹配指定前缀的命名空间 URI。
lookupPrefix()	返回匹配指定命名空间 URI 的前缀。
normalize()	把节点（包括属性）下的所有文本节点放置到一个"标准"的格式中，其中只有结构（比如元素、注释、处理指令、CDATA 区段以及实体引用）来分隔 Text 节点，例如，既没有相邻的 Text 节点，也没有空的 Text 节点。
removeChild()	删除子节点。
replaceChild()	替换子节点。
setUserData(key,data,handler)	把对象关联到节点上的键。

XML DOM - NodeList 对象

NodeList 对象代表一个有序的节点列表。

NodeList 对象

节点列表中的节点可以通过其对应的索引数字（从 0 开始计数）进行访问。

节点列表可保持其自身的更新。如果节点列表或 XML 文档中的某个元素被删除或添加，列表也会被自动更新。

注意：在一个节点列表中，节点被返回的顺序与它们在 XML 文档中被规定的顺序相同。

NodeList 对象属性

属性	描述
length	返回节点列表中的节点数量。

NodeList 对象方法

方法	描述
item()	返回节点列表中指定索引号的节点。

XML DOM - NamedNodeMap 对象

NamedNodeMap 对象代表一个节点的无序列表。

NamedNodeMap 对象

NamedNodeMap 中的节点可以通过它们的名称进行访问。

NamedNodeMap 将会自我更新。如果在节点列表或 XML 文档中删除或添加一个元素，那么该列表将会自动更新。

注意：在命名节点图中，节点不会以任何特定的顺序返回。

NamedNodeMap 对象属性

属性	描述
length	返回列表中节点的数量。

NamedNodeMap 对象方法

方法	描述
getNamedItem()	返回指定的节点（通过名称）。
getNamedItemNS()	返回指定的节点（通过名称和命名空间）。
item()	返回指定索引号的节点。
removeNamedItem()	删除指定的节点（通过名称）。
removeNamedItemNS()	删除指定的节点（通过名称和命名空间）。
setNamedItem()	设置指定的节点（通过名称）。
setNamedItemNS()	设置指定的节点（通过名称和命名空间）。

XML DOM - Document 对象

Document 对象代表整个 XML 文档。

Document 对象

Document 对象是文档树的根，并为我们提供对文档数据的最初（或最顶层）的访问入口。

由于元素节点、文本节点、注释、处理指令等均无法存在于文档之外，Document 对象也提供了创建这些对象的方法。Node 对象提供了一个 ownerDocument 属性，此属性可把它们与在其中创建它们的 Document 关联起来。

Document 对象属性

属性	描述
async	规定 XML 文件的下载是否应当被异步处理。
childNodes	返回文档的子节点的节点列表。
doctype	返回与文档相关的文档类型声明（DTD，全称 Document Type Declaration）。
documentElement	返回文档的根节点。
documentURI	设置或返回文档的位置。
domConfig	返回 normalizeDocument() 被调用时所使用的配置。
firstChild	返回文档的第一个子节点。
implementation	返回处理该文档的 DOMImplementation 对象。
inputEncoding	返回用于文档的编码方式（在解析时）。
lastChild	返回文档的最后一个子节点。
nodeName	返回节点的名称（根据节点的类型）。
nodeType	返回节点的节点类型。
nodeValue	设置或返回节点的值（根据节点的类型）。
strictErrorChecking	设置或返回是否强制进行错误检查。
xmlEncoding	返回文档的 XML 编码。
xmlStandalone	设置或返回文档是否为 standalone。
xmlVersion	设置或返回文档的 XML 版本。

Document 对象方法

方法	描述
adoptNode(sourcenode)	从另一个文档向本文档选定一个节点，然后返回被选节点。
createAttribute(name)	创建带有指定名称的属性节点，并返回新的 Attr 对象。
createAttributeNS(uri,name)	创建带有指定名称和命名空间的属性节点，并返回新的 Attr 对象。
createCDATASection()	创建 CDATA 区段节点。
createComment()	创建注释节点。
createDocumentFragment()	创建空的 DocumentFragment 对象，并返回此对象。
createElement()	创建元素节点。
createElementNS()	创建带有指定命名空间的元素节点。
createEntityReference(name)	创建 EntityReference 对象，并返回此对象。
createProcessingInstruction(target,data)	创建一个 ProcessingInstruction 对象，并返回此对象。
createTextNode()	创建文本节点。
getElementById(id)	返回带有指定值的 ID 属性的元素。如果不存在这样的元素，则返回 null。
getElementsByTagName()	返回带有指定名称的所有元素的 NodeList。
getElementsByTagNameNS()	返回带有指定名称和命名空间的所有元素的 NodeList。
importNode(nodetoimport,deep)	从另一个文档向本文档选定一个节点。该方法创建源节点的一个新的副本。如果 deep 参数设置为 true，它将导入指定节点的所有子节点。 如果设置为 false，它将只导入节点本身。该方法返回被导入的节点。
normalizeDocument()	
renameNode()	重命名元素或属性节点。

XML DOM - DocumentImplementation 对象

DocumentImplementation 对象

DOMImplementation 对象执行的操作是独立于文档对象模型的任何特定实例。

DocumentImplementation 对象方法

方法	描述
createDocument(nsURI, name, doctype)	创建一个新的指定文档类型的 DOM Document 对象。
createDocumentType(name, pubId, systemId)	创建一个空的 DocumentType 节点。
getFeature(feature, version)	返回一个对象，此对象可执行带有指定特性和版本的 API。
hasFeature(feature, version)	检查 DOM implementation 是否实现指定的特性和版本。

XML DOM - DocumentType 对象

DocumentType 对象

每个文档都包含一个 DOCTYPE 属性，该属性值可以是一个空值或是一个 DocumentType 对象。

DocumentType 对象提供了一个接口，用于定义 XML 文档的实体。

DocumentType 对象属性

属性	描述
entities	返回包含有在 DTD 中所声明的实体的 NamedNodeMap。
internalSubset	以字符串形式返回内部 DTD。
name	返回 DTD 的名称。
notations	返回包含 DTD 声明的符号的 NamedNodeMap。
systemId	返回外部 DTD 的系统标识符。

XML DOM - ProcessingInstruction 对象

ProcessingInstruction 对象

ProcessingInstruction 对象代表一个处理指令。

处理指令用于维护 XML 文档的文本中特定处理器的信息。

ProcessingInstruction 对象属性

属性	描述
data	设置或返回处理指令的内容。
target	返回处理指令的目标。

XML DOM - Element 对象

Element 对象

Element 对象代表 XML 文档中的一个元素。元素可以包含属性、其他元素或文本。如果一个元素包含文本，则在文本节点中表示该文本。

重要事项：文本永远存储在文本节点中。在 DOM 处理过程中的一个常见的错误是，导航到元素节点，并认为此节点含有文本。不过，即使最简单的元素节点之下也拥有文本节点。举例，在 <year>2005</year> 中，有一个元素节点（year），同时此节点之下存在一个文本节点，其中含有文本（2005）。

由于 Element 对象也是一种节点，因此它可继承 Node 对象的属性和方法。

Element 对象属性

属性	描述
attributes	返回元素的属性的 NamedNodeMap。
baseURI	返回元素的绝对基准 URI。
childNodes	返回元素的子节点的 NodeList。
firstChild	返回元素的第一个子节点。
lastChild	返回元素的最后一个子节点。
localName	返回元素名称的本地部分。
namespaceURI	返回元素的命名空间 URI。
nextSibling	返回元素之后紧接的节点。
nodeName	返回节点的名称，根据其类型。
nodeType	返回节点的类型。
ownerDocument	返回元素所属的根元素 (document 对象)。
parentNode	返回元素的父节点。
prefix	设置或返回元素的命名空间前缀。
previousSibling	返回元素之前紧接的节点。
schemaTypeInfo	返回与元素相关联的类型信息。
tagName	返回元素的名称。
textContent	设置或返回元素及其后代的文本内容。

Element 对象方法

方法	描述
appendChild()	把新的子节点添加到节点的子节点列表末尾。
cloneNode()	克隆节点。
compareDocumentPosition()	比较两个节点的文档位置。
getAttribute()	返回属性的值。
getAttributeNS()	返回属性的值（带有命名空间）。
getAttributeNode()	以 Attribute 对象返回属性节点。
getAttributeNodeNS()	以 Attribute 对象返回属性节点（带有命名空间）。
getElementsByTagName()	返回匹配的元素节点及它们的子节点的 NodeList。
getElementsByTagNameNS()	返回匹配的元素节点（带有命名空间）及它们的子节点的 NodeList。

getFeature(feature,version)	返回 DOM 对象，此对象可执行带有指定特性和版本的专门的 API。
getUserData(key)	返回与节点上键关联的对象。此对象必须首先通过使用相同的键调用 setUserData 来设置到此节点。
hasAttribute()	返回元素是否拥有匹配指定名称的属性。
hasAttributeNS()	返回元素是否拥有匹配指定名称和命名空间的属性。
hasAttributes()	返回元素是否拥有属性。
hasChildNodes()	返回元素是否拥有子节点。
insertBefore()	在已有的子节点之前插入一个新的子节点。
isDefaultNamespace(URI)	返回指定的 namespaceURI 是否为默认。
isEqualNode()	检查两个节点是否相等。
isSameNode()	检查两个节点是否为同一节点。
isSupported(feature,version)	返回指定的特性是否在此元素上得到支持。
lookupNamespaceURI()	返回匹配指定前缀的命名空间 URI。
lookupPrefix()	返回匹配指定命名空间 URI 的前缀。
normalize()	把节点（包括属性）下的所有文本节点放置到一个"标准"的格式中，其中只有结构（比如元素、注释、处理指令、CDATA 区段以及实体引用）来分隔 Text 节点，例如，既没有相邻的 Text 节点，也没有空的 Text 节点。
removeAttribute()	删除指定的属性。
removeAttributeNS()	删除指定的属性（带有命名空间）。
removeAttributeNode()	删除指定的属性节点。
removeChild()	删除子节点。
replaceChild()	替换子节点。
setUserData(key,data,handler)	把对象关联到元素上的键。
setAttribute()	添加新属性。
setAttributeNS()	添加新属性（带有命名空间）。
setAttributeNode()	添加新的属性节点。
setAttributeNodeNS(attrnode)	添加新的属性节点（带有命名空间）。
setIdAttribute(name,isId)	如果 Attribute 对象的 isId 属性为 true，那么此方法会把指定的属性声明为一个用户确定 ID 的属性（user-determined ID attribute）。
setIdAttributeNS(uri,name,isId)	如果 Attribute 对象的 isId 属性为 true，那么此方法会把指定的属性声明为一个用户确定 ID 的属性（user-determined ID attribute）（带有命名空间）。

setIdAttributeNode(idAttr,isId)	如果 Attribute 对象的 isId 属性为 true，那么此方法会把指定的属性声明为一个用户确定 ID 的属性（user-determined ID attribute）。

XML DOM - Attr 对象

Attr 对象

Attr 对象表示 Element 对象的属性。属性的容许值通常定义在 DTD 中。

由于 Attr 对象也是一种节点，因此它继承 Node 对象的属性和方法。不过属性无法拥有父节点，同时属性也不被认为是元素的子节点，对于许多 Node 属性来说都将返回 null。

Attr 对象属性

属性	描述
baseURI	返回属性的绝对基准 URI。
isId	如果属性是 ID 类型，则返回 true，否则返回 false。
localName	返回属性名称的本地部分。
name	返回属性的名称。
namespaceURI	返回属性的命名空间 URI。
nodeName	返回节点的名称，根据其类型。
nodeType	返回节点的类型。
nodeValue	设置或返回节点的值，根据其类型。
ownerDocument	返回属性所属的根元素（document 对象）。
ownerElement	返回属性所附属的元素节点。
prefix	设置或返回属性的命名空间前缀。
schemaTypeInfo	返回与属性相关联的类型信息。
specified	如果属性值被设置在文档中，则返回 true，如果其默认值被设置在 DTD/Schema 中，则返回 false。
textContent	设置或返回属性的文本内容。
value	设置或返回属性的值。

XML DOM - Text 对象

Text 对象

Text 对象表示元素或属性的文本内容。

Text 对象属性

属性	描述
data	设置或返回元素或属性的文本。
isElementContentWhitespace	判断文本节点是否包含空白字符内容。如果文本节点包含空白字符内容，则返回 true，否则返回 false。
length	返回元素或属性的文本长度。
wholeText	以文档中的顺序向此节点返回相邻文本节点的所有文本。

Text 对象方法

方法	描述
appendData()	向节点追加数据。
deleteData()	从节点删除数据。
insertData()	向节点中插入数据。
replaceData()	替换节点中的数据。
replaceWholeText(text)	使用指定文本来替换此节点以及所有相邻的文本节点。
splitText()	在指定的偏移处将此节点拆分为两个节点，同时返回包含偏移处之后的文本的新节点。
substringData()	从节点提取数据。

XML DOM - CDATASection 对象

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 外部 JavaScript 用于加载 XML 文件。

createCDATASection() - 创建一个 CDATA 区段节点

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("book");

newtext="Special Offer & Book Sale";

for (i=0;i<x.length;i++)
{
newCDATA=xmlDoc.createCDATASection(newtext);
x[i].appendChild(newCDATA);
}

for (i=0;i<x.length;i++)
{
document.write(x[i].getElementsByTagName("title")[0].childNodes[0].nodeValue);
document.write(" - ");
document.write(x[i].lastChild.nodeValue);
document.write("<br>");
}
</script>
</body>
</html>
```

CDATASection 对象

CDATASection 对象表示文档中的 CDATA 区段。

CDATA 区段包含了不会被解析器解析的文本。一个 CDATA 区段中的标签不会被视为标记，同时实体也不会被展开。主要的目的是为了包含诸如 XML 片段之类的材料，而无需转义所有的分隔符。

在一个 CDATA 区段中唯一被识别的分隔符是 "]]>"，它可标示 CDATA 区段的结束。CDATA 区段不能进行嵌套。

CDATASection 对象属性

属性	描述
data	设置或返回此节点的文本。
length	返回 CDATA 区段的长度。

CDATASection 对象方法

方法	描述
appendData()	向节点追加数据。
deleteData()	从节点删除数据。
insertData()	向节点中插入数据。
replaceData()	替换节点中的数据。
splitText()	把 CDATA 节点分拆为两个节点。
substringData()	从节点提取数据。

XML DOM - Comment 对象

尝试一下 - 实例

下面的实例使用 XML 文件 books.xml。 外部 JavaScript 用于加载 XML 文件。

createComment() - 创建一个注释节点

```
<!DOCTYPE html>
<html>
<head>
<script src="loadxmldoc.js"></script>
</head>
<body>

<script>
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName('book');

for (i=0;i<x.length;i++)
{
newComment=xmlDoc.createComment("Revised April 2008");
x[i].appendChild(newComment);
}

for (i=0;i<x.length;i++)
{
document.write(x[i].getElementsByTagName("title")[0].childNodes[0].nodeValue);
document.write(" - ");
document.write(x[i].lastChild.nodeValue);
document.write("<br>");
}
</script>
</body>
</html>
```

Comment 对象

Comment 对象表示文档中注释节点的内容。

Comment 对象属性

属性	描述
data	设置或返回此节点的文本。
length	返回此节点的文本的长度。

Comment 对象方法

方法	描述
appendData()	向节点追加数据。
deleteData()	从节点删除数据。
insertData()	向节点中插入数据。
replaceData()	替换节点中的数据。
substringData()	从节点中提取数据。

XMLHttpRequest 对象

通过 XMLHttpRequest 对象，您可以在不重新加载整个页面的情况下更新网页中的某个部分。

尝试一下 - 实例

一个简单的 XMLHttpRequest 实例 创建一个简单的 XMLHttpRequest，从 TXT 文件中检索数据。

```
<!DOCTYPE html>
<html>
<head>
<script>
function loadXMLDoc()
{
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","xmlhttp_info.txt",true);
xmlhttp.send();
}
</script>
</head>
<body>

<h2>Using the XMLHttpRequest object</h2>
<div id="myDiv"></div>
<button type="button" onclick="loadXMLDoc()">Change Content</button>

</body>
</html>
```

通过 getAllResponseHeaders() 检索头信息 检索资源（文件）的头信息。

```
<!DOCTYPE html>
<html>
<head>
<script>
function loadXMLDoc(url)
{
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById('p1').innerHTML=xmlhttp.getAllResponseHeaders();
    }
  }
xmlhttp.open("GET",url,true);
xmlhttp.send();
}
</script>
</head>
<body>

<p id="p1">The getAllResponseHeaders() function returns the header information of a resou
<button onclick="loadXMLDoc('xmlhttp_info.txt')">Get header information</button>

</body>
</html>
```

通过 getResponseHeader() 检索指定头信息 检索资源（文件）的指定头信息。

```
<!DOCTYPE html>
<html>
<head>
<script>
function loadXMLDoc(url)
{
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById('p1').innerHTML="Last modified: " + xmlhttp.getResponseHeader
    }
  }
xmlhttp.open("GET",url,true);
xmlhttp.send();
}
</script>
</head>
<body>

<p id="p1">The getResponseHeader() function is used to return specific header information
<button onclick="loadXMLDoc('xmlhttp_info.txt')">Get "Last-Modified" information</button>

</body>
</html>
```

检索 ASP 文件的内容 当用户在输入字段键入字符时，网页如何与 Web 服务器进行通信。

```
<!DOCTYPE html>
<html>
<head>
<script>
function showHint(str)
{
if (str.length==0)
  {
  document.getElementById("txtHint").innerHTML="";
  return;
  }
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","gethint.php?q="+str,true);
xmlhttp.send();
}
</script>
</head>
<body>

<h3>Start typing a name in the input field below:</h3>
<form action="">
First name: <input type="text" id="txt1" onkeyup="showHint(this.value)" />
</form>
<p>Suggestions: <span id="txtHint"></span></p>

</body>
</html>
```

从数据库中检索内容 网页如何通过 XMLHttpRequest 对象从数据库中提取信息。

```
<!DOCTYPE html>
<html>
<head>
<script>
function showCustomer(str)
{
if (str=="")
  {
  document.getElementById("txtHint").innerHTML="";
  return;
  }
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","getcustomer.php?q="+str,true);
xmlhttp.send();
}
</script>
</head>
<body>

<form action="">
<select name="customers" onchange="showCustomer(this.value)">
<option value="">Select a customer:</option>
<option value="ALFKI">Alfreds Futterkiste</option>
<option value="NORTS ">North/South</option>
<option value="WOLZA">Wolski Zajazd</option>
</select>
</form>
<br>
<div id="txtHint">Customer info will be listed here...</div>

</body>
</html>
```

检索 XML 文件的内容 创建一个 XMLHttpRequest 从 XML 文件中检索数据并把数据显示在一个 HTML 表格中。

```
<!DOCTYPE html>
<html>
<head>
<script>
function loadXMLDoc(url)
{
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    txt="<table border='1'><tr><th>Title</th><th>Artist</th></tr>";
    x=xmlhttp.responseXML.documentElement.getElementsByTagName("CD");
    for (i=0;i<x.length;i++)
      {
      txt=txt + "<tr>";
      xx=x[i].getElementsByTagName("TITLE");
        {
        try
          {
          txt=txt + "<td>" + xx[0].firstChild.nodeValue + "</td>";
          }
        catch (er)
          {
          txt=txt + "<td> </td>";
          }
        }
      xx=x[i].getElementsByTagName("ARTIST");
        {
        try
          {
          txt=txt + "<td>" + xx[0].firstChild.nodeValue + "</td>";
          }
        catch (er)
          {
          txt=txt + "<td> </td>";
          }
        }
      txt=txt + "</tr>";
      }
    txt=txt + "</table>";
    document.getElementById('txtCDInfo').innerHTML=txt;
    }
  }
xmlhttp.open("GET",url,true);
xmlhttp.send();
}
</script>
</head>
<body>

<div id="txtCDInfo">
<button onclick="loadXMLDoc('cd_catalog.xml')">Get CD info</button>
</div>

</body>
</html>
```

XMLHttpRequest 对象

XMLHttpRequest 对象是用于幕后与服务器交换数据。

XMLHttpRequest 对象是开发者的梦想，因为您可以：

- 在不重新加载页面的情况下更新网页
- 在页面已加载后从服务器请求数据
- 在页面已加载后从服务器接收数据
- 在后台向服务器发送数据

XMLHttpRequest 对象方法

方法	描述
abort()	取消当前的请求。
getAllResponseHeaders()	返回头信息。
getResponseHeader()	返回指定的头信息。
open(method,url,async,uname,pswd)	规定请求的类型，URL，请求是否应该进行异步处理，以及请求的其他可选属性。

method：请求的类型：GET 或 POST url：文件在服务器上的位置 async：true（异步）或 false（同步） | | send(string) | 发送请求到服务器。

string：仅用于 POST 请求 | | setRequestHeader() | 把标签/值对添加到要发送的头文件。 |

XMLHttpRequest 对象属性

属性	描述
onreadystatechange	存储函数（或函数的名称）在每次 readyState 属性变化时被自动调用。
readyState	存放了 XMLHttpRequest 的状态。从 0 到 4 变化：

0：请求未初始化 1：服务器建立连接 2：收到的请求 3：处理请求 4：请求完成和响应准备就绪 | | responseText | 返回作为一个字符串的响应数据。 | | responseXML | 返回作为 XML 数据响应数据。 | | status | 返回状态数（例如 "404" 为 "Not Found" 或 "200" 为 "OK"）。 | | statusText | 返回状态文本（如 "Not Found" 或 "OK"）。 |

XML DOM Parse Error 对象

微软的 parseError 对象可用于从微软的 XML 分析器取回错误信息。

要查看 Firefox 如何处理解析器错误，请看本教程的下一个页面。

parseError 对象

在您试图打开一个 XML 文档时，就可能发生一个解析器错误（parser-error）。

通过这个 parseError 对象，您可取回错误代码、错误文本、引起错误的行等等。

注意：parseError 对象不属于 W3C DOM 标准！

文件错误（File Error）

在下面的代码中，我们会试图加载一个不存在的文件，并显示某些错误属性：

实例

```
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.load("ksdjf.xml");

document.write("Error code: " + xmlDoc.parseError.errorCode);
document.write("
Error reason: " + xmlDoc.parseError.reason);
document.write("
Error line: " + xmlDoc.parseError.line);
```

XML 错误（XML Error）

在下面的代码中，我们会让解析器加载一个形式不良的 XML 文档。

（您可以在我们的 XML 教程中阅读更多有关形式良好且有效的 XML。）

实例

```
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.load("note_error.xml");

document.write("Error code: " + xmlDoc.parseError.errorCode);
document.write("
Error reason: " + xmlDoc.parseError.reason);
document.write("
Error line: " + xmlDoc.parseError.line);
```

查看 XML 文件：note_error.xml

parseError 对象的属性

属性	描述
errorCode	返回一个长整数错误代码。
reason	返回一个字符串，包含错误的原因。
line	返回一个长整数，代表错误的行号。
linepos	返回一个长整数，代表错误的行位置。
srcText	返回一个字符串，包含引起错误的行。
url	返回指向被加载文档的 URL。
filepos	返回错误的一个长整型文件位置。

XML DOM 解析器错误

当 Firefox 遇到解析器错误，它会载入一个包含错误的 XML 文档。

在 Firefox 中的解析器错误

在您试图打开一个 XML 文档时，就可能发生一个解析器错误（parser-error）。

与 Internet Explorer 浏览器不同，如果 Firefox 遇到错误，它会载入包含错误描述的 XML 文档中。

XML 错误文档的根节点的名称是 "parsererror"。这是用来检查是否有错误。

XML 错误（XML Error）

在下面的代码中，我们会让解析器加载一个形式不良的 XML 文档。

（您可以在我们的 XML 教程中阅读更多有关形式良好且有效的 XML。）

实例

```
xmlDoc=document.implementation.createDocument("","",null);
xmlDoc.async=false;
xmlDoc.load("note_error.xml");

if (xmlDoc.documentElement.nodeName=="parsererror")
{
errStr=xmlDoc.documentElement.childNodes[0].nodeValue;
errStr=errStr.replace(/</g, "&lt;");
document.write(errStr);
}
else
{
document.write("XML is valid");
}
```

查看 XML 文件：note_error.xml

实例解释：

1. 加载 XML 文件
2. 检查根节点的节点名称是否是 "parsererror"
3. 把错误字符串载入变量 "errStr"
4. 在错误字符串编写为 HTML 之前，把 "<" 字符替换为 "<"

注意：实际上，只有 Internet Explorer 会用 DTD 检查您的 XML，Firefox 不会。

跨浏览器的错误检查

在这里，我们创建了一个 XML 加载函数，在 Internet Explorer 和 Firefox 中检查解析器错误：

实例

```
function loadXMLDocErr(dname)
{
try //Internet Explorer
{
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.load(dname);

if (xmlDoc.parseError.errorCode != 0)
{
alert("Error in line " + xmlDoc.parseError.line +
" position " + xmlDoc.parseError.linePos +
"nError Code: " + xmlDoc.parseError.errorCode +
"nError Reason: " + xmlDoc.parseError.reason +
"Error Line: " + xmlDoc.parseError.srcText);
return(null);
}
}
catch(e)
{
try //Firefox
{
xmlDoc=document.implementation.createDocument("","",null);
xmlDoc.async=false;
xmlDoc.load(dname);
if (xmlDoc.documentElement.nodeName=="parsererror")
{
alert(xmlDoc.documentElement.childNodes[0].nodeValue);
return(null);
}
}
catch(e) {alert(e.message)}
}
try
{
return(xmlDoc);
}
catch(e) {alert(e.message)}
return(null);
}
```

查看 XML 文件：note_error.xml

实例解释 - Internet Explorer：

1. 第一行创建一个空的微软 XML 文档对象。
2. 第二行关闭异步加载，确保在文档完全加载之前解析器不会继续执行脚本。
3. 第三行告知解析器加载名为 "note_error.xml" 的 XML 文档。

4. 如果 parseError 对象的 ErrorCode 属性和 "0" 不同，提醒错误并退出函数。
5. 如果 ErrorCode 属性为 "0"，返回 XML 文档。

实例解释 - Firefox：

1. 第一行创建一个空的XML文档对象。
2. 第二行关闭异步加载，确保在文档完全加载之前解析器不会继续执行脚本。
3. 第三行告知解析器加载名为 "note_error.xml" 的 XML 文档。
4. 如果返回的文档是一个错误的文档，提醒错误并退出函数。
5. 如果没有，则返回 XML 文档。

XSLT 教程

XSL 语言

它起始于 XSL，结束于 XSLT、XPath 以及 XSL-FO。

起始于 XSL

XSL 指扩展样式表语言（E**X**tensible **S**tylesheet **L**anguage）。

万维网联盟（W3C）开始发展 XSL 的原因是：存在着对于基于 XML 的样式表语言的需求。

CSS = HTML 样式表

HTML 使用预先定义的标签，每个标签的意义很容易被理解。

HTML 中的 <table> 标签定义表格 - 并且浏览器清楚如何显示它。

向 HTML 元素添加样式是很容易的。通过 CSS，很容易告知浏览器用特定的字体或颜色显示一个元素。

XSL = XML 样式表

XML 不使用预先定义的标签（我们可以使用任何喜欢的标签名），并且每个标签的意义并不都那么容易被理解。

<table> 标签意味着一个 HTML 表格，一件家具，或是别的什么东西 - 浏览器不清楚如何显示它。

XSL 可描述如何来显示 XML 文档！

XSL - 不仅仅是样式表语言

XSL 包括三部分：

- XSLT - 一种用于转换 XML 文档的语言。
- XPath - 一种用于在 XML 文档中导航的语言。
- XSL-FO - 一种用于格式化 XML 文档的语言。

本教程的主要内容是 XSLT

本教程的其余部分是 XSLT - 用来转换 XML 文档的语言。

如需学习更多有关 XPath 和 XSL-FO 的知识，请访问我们的XPath 教程 和 XSL-FO 教程。

XSLT 简介

XSLT 是一种用于将 XML 文档转换为 XHTML 文档或其他 XML 文档的语言。

XPath 是一种用于在 XML 文档中进行导航的语言。

您需要具备的基础知识

在您继续学习之前，需要对以下知识有基本的了解：

- HTML / XHTML
- XML / XML 命名空间
- XPath

如果您想要首先学习这些项目，请在我们的首页访问这些教程。

什么是 XSLT？

- XSLT 指 XSL 转换（XSL Transformations）
- XSLT 是 XSL 中最重要的部分
- XSLT 可将一种 XML 文档转换为另外一种 XML 文档
- XSLT 使用 XPath 在 XML 文档中进行导航
- XSLT 是一个 W3C 标准

XSLT = XSL 转换

XSLT 是 XSL 中最重要的部分。

XSLT 用于将一种 XML 文档转换为另外一种 XML 文档，或者可被浏览器识别的其他类型的文档，比如 HTML 和 XHTML。通常，XSLT 是通过把每个 XML 元素转换为 (X)HTML 元素来完成这项工作的。

通过 XSLT，您可以向输出文件添加元素和属性，或从输出文件移除元素和属性。您也可重新排列并分类元素，执行测试并决定隐藏或显示哪个元素，等等。

描述转化过程的一种通常的说法是，**XSLT** 把 **XML** 源树转换为 **XML** 结果树。

XSLT 使用 XPath

XSLT 使用 XPath 在 XML 文档中查找信息。XPath 被用来通过元素和属性在 XML 文档中进行导航。

如果您想要首先学习 XPath，请访问我们的 XPath 教程。

它如何工作？

在转换过程中，XSLT 使用 XPath 来定义源文档中可匹配一个或多个预定义模板的部分。一旦匹配被找到，XSLT 就会把源文档的匹配部分转换为结果文档。

XSLT 是一个 W3C 标准

XSLT 在 1999 年 11 月 16 日被确立为 W3C 标准。

如需更多有关 W3C 的 XSLT 活动的信息，请访问我们的 W3C 教程。

XSLT 浏览器

所有主流的浏览器都支持 XML 和 XSLT。

Mozilla Firefox

从版本 3 开始，Firefox 就已支持 XML、XSLT 和 XPath。

Internet Explorer

从版本 6 开始，Internet Explorer 就已支持 XML、XSLT 和 XPath。

Internet Explorer 5 不兼容官方的 W3C XSL 标准。

Google Chrome

从版本 1 开始，Chrome 就已支持 XML、XSLT 和 XPath。

Opera

从版本 9 开始，Opera 就已支持 XML、XSLT 和 XPath。Opera 8 仅支持 XML + CSS。

Apple Safari

从版本 3 开始，Safari 就已支持 XML 和 XSLT。

XSLT - 转换

实例研究：如何使用 XSLT 将 XML 转换为 XHTML。

我们会在下一章对本实例的细节进行解释。

正确的样式表声明

把文档声明为 XSL 样式表的根元素是 <xsl:stylesheet> 或 <xsl:transform>。

注意：<xsl:stylesheet> 和 <xsl:transform> 是完全同义的，均可被使用！

根据 W3C 的 XSLT 标准，声明 XSL 样式表的正确方法是：

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

或者：

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

如需访问 XSLT 的元素、属性以及特性，我们必须在文档顶端声明 XSLT 命名空间。

xmlns:xsl="http://www.w3.org/1999/XSL/Transform" 指向了官方的 W3C XSLT 命名空间。如果您使用此命名空间，就必须包含属性 version="1.0"。

从一个原始的 XML 文档开始

我们现在要把下面这个 XML 文档（"cdcatalog.xml"）转换为 XHTML：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
.
.
</catalog>
```

在 **Firefox** 和 **Internet Explorer** 中查看 **XML** 文件：打开 XML 文件（通常通过点击某个链接） - XML 文档会以颜色化的代码方式来显示根元素及子元素。点击元素左侧的加号（+）或减号（-）可展开或收缩元素的结构。如需查看原始的 XML 源文件（不带有加号和减号），请在浏览器菜单中选择"查看页面源代码"或"查看源代码"。

在 **Netscape 6** 中查看 **XML** 文件：打开 XML 文件，然后在 XML 文件中右击，并选择"查看页面源代码"。XML 文档会以颜色化的代码方式来显示根元素及子元素。

在 **Opera 7** 中查看 **XML** 文件：打开 XML 文件，然后在 XML 文件中右击，选择"框架"/"查看源代码"。XML 文档将显示为纯文本。

查看 "cdcatalog.xml"

创建 XSL 样式表

然后创建一个带有转换模板的 XSL 样式表（"cdcatalog.xsl"）：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

查看 "cdcatalog.xsl"

把 XSL 样式表链接到 XML 文档

向 XML 文档（"cdcatalog.xml"）添加 XSL 样式表引用：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
.
.
</catalog>
```

如果您使用的浏览器兼容 XSLT，它会很顺利地把您的 XML 转换为 XHTML。

查看结果

我们会在下一章对上面的例子中的细节进行解释。

XSLT <xsl:template> 元素

XSL 样式表由一个或多套被称为模板（template）的规则组成。

每个模板含有当某个指定的节点被匹配时所应用的规则。

<xsl:template> 元素

<xsl:template> 元素用于构建模板。

match 属性用于关联 XML 元素和模板。match 属性也可用来为整个 XML 文档定义模板。match 属性的值是 XPath 表达式（举例，match="/" 定义整个文档）。

好了，让我们看一下上一章中的 XSL 文件的简化版本：

实例

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

实例解释

由于 XSL 样式表本身也是一个 XML 文档，因此它总是由 XML 声明起始：**<?xml version="1.0" encoding="ISO-8859-1"?>**.

下一个元素，**<xsl:stylesheet>**，，定义此文档是一个 XSLT 样式表文档（连同版本号和 XSLT 命名空间属性）。

<xsl:template> 元素定义了一个模板。而 **match="/"** 属性则把此模板与 XML 源文档的根相联系。

<xsl:template> 元素内部的内容定义了写到输出结果的 HTML 代码。

最后两行定义了模板的结尾及样式表的结尾。

这个实例的结果有一点小缺陷，因为数据没有从 XML 文档被复制到输出。在下一章中，您将学习到如何使用 **<xsl:value-of>** 元素从 XML 元素选取值。

XSLT <xsl:value-of> 元素

<xsl:value-of> 元素用于提取某个选定节点的值。

<xsl:value-of> 元素

<xsl:value-of> 元素用于提取某个 XML 元素的值，并把值添加到转换的输出流中：

实例

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<tr>
<td><xsl:value-of select="catalog/cd/title"/></td>
<td><xsl:value-of select="catalog/cd/artist"/></td>
</tr>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

实例解释

注意：在上面的实例中，**select** 属性的值是一个 XPath 表达式。这个 XPath 表达式的工作方式类似于定位某个文件系统，在其中正斜杠（/）可选择子目录。

上面实例的结果有一点小缺陷，仅有一行数据从 XML 文档被复制到输出。在下一章中，您将学习到如何使用 **<xsl:for-each>** 元素来循环遍历 XML 元素，并显示所有的记录。

XSLT <xsl:for-each> 元素

<xsl:for-each> 元素允许您在 XSLT 中进行循环。

<xsl:for-each> 元素

XSL <xsl:for-each> 元素可用于选取指定的节点集中的每个 XML 元素：

实例

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

注意：**select** 属性的值是一个 XPath 表达式。这个 XPath 表达式的工作方式类似于定位某个文件系统，在其中正斜杠（/）可选择子目录。

过滤输出结果

通过在 <xsl:for-each> 元素中添加一个选择属性的判别式，我们也可以过滤从 XML 文件输出的结果。

<xsl:for-each select="catalog/cd[artist='Bob Dylan']">

合法的过滤运算符:

- = （等于）

- != （不等于）
- < （小于）
- > （大于）

看看调整后的 XSL 样式表：

实例

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="catalog/cd[artist='Bob Dylan']">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

XSLT <xsl:sort> 元素

<xsl:sort> 元素用于对输出结果进行排序。

在何处放置排序信息

如需对输出结果进行排序，只要简单地在 XSL 文件中的 <xsl:for-each> 元素内部添加一个 <xsl:sort> 元素：

实例

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
<xsl:sort select="artist"/>
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

注意：**select** 属性指示需要排序的 XML 元素。

XSLT <xsl:if> 元素

<xsl:if> 元素用于放置针对 XML 文件内容的条件测试。

<xsl:if> 元素

如需放置针对 XML 文件内容的条件测试，请向 XSL 文档添加 <xsl:if> 元素。

语法

```
<xsl:if test="_expression_">
...如果条件成立则输出...
</xsl:if>
```

在何处放置 <xsl:if> 元素

如需添加有条件的测试，请在 XSL 文件中的 <xsl:for-each> 元素内部添加 <xsl:if> 元素：

实例

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
<xsl:if test="price &gt; 10">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:if>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

注意：必需的 **test** 属性的值包含了需要求值的表达式。

上面的代码仅仅会输出价格高于 10 的 CD 的 title 和 artist 元素。

XSLT <xsl:choose> 元素

<xsl:choose> 元素用于结合 <xsl:when> 和 <xsl:otherwise> 来表达多重条件测试。

<xsl:choose> 元素

语法

```
<xsl:choose>
<xsl:when test="_expression_">
... some output ...
</xsl:when>
<xsl:otherwise>
... some output ....
</xsl:otherwise>
</xsl:choose>
```

在何处放置选择条件

如需插入针对 XML 文件的多重条件测试，请向 XSL 文件添加 <xsl:choose>、<xsl:when> 以及 <xsl:otherwise> 元素：

实例

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
<tr>
<td><xsl:value-of select="title"/></td>
<xsl:choose>
<xsl:when test="price &gt; 10">
<td bgcolor="#ff00ff">
<xsl:value-of select="artist"/></td>
</xsl:when>
<xsl:otherwise>
<td><xsl:value-of select="artist"/></td>
</xsl:otherwise>
</xsl:choose>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

上面的代码会在 CD 的价格高于 10 时向 "Artist" 列添加粉色的背景颜色。

另一个实例

这是另外一个包含两个 <xsl:when> 元素的实例：

实例

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
<tr>
<td><xsl:value-of select="title"/></td>
<xsl:choose>
<xsl:when test="price &gt; 10">
<td bgcolor="#ff00ff">
<xsl:value-of select="artist"/></td>
</xsl:when>
<xsl:when test="price &gt; 9">
<td bgcolor="#cccccc">
<xsl:value-of select="artist"/></td>
</xsl:when>
<xsl:otherwise>
<td><xsl:value-of select="artist"/></td>
</xsl:otherwise>
</xsl:choose>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

上面的代码会在 CD 的价格高于 10 时向 "Artist" 列添加粉色的背景颜色，并在 CD 的价格高于 9 且低于等于 10 时向 "Artist" 列添加灰色的背景颜色。

XSLT <xsl:apply-templates> 元素

<xsl:apply-templates> 元素可把一个模板应用于当前的元素或者当前元素的子节点。

<xsl:apply-templates> 元素

<xsl:apply-templates> 元素可把一个模板应用于当前的元素或者当前元素的子节点。

假如我们向 <xsl:apply-templates> 元素添加一个 select 属性，此元素就会仅仅处理与属性值匹配的子元素。我们可以使用 select 属性来规定子节点被处理的顺序。

请看下面的 XSL 样式表：

实例

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>

<xsl:template match="cd">
<p>
<xsl:apply-templates select="title"/>
<xsl:apply-templates select="artist"/>
</p>
</xsl:template>

<xsl:template match="title">
Title: <span style="color:#ff0000">
<xsl:value-of select="."/></span>
<br />
</xsl:template>

<xsl:template match="artist">
Artist: <span style="color:#00ff00">
<xsl:value-of select="."/></span>
<br />
</xsl:template>

</xsl:stylesheet>
```

XSLT - 在客户端

如果您的浏览器支持 XSLT，那么在浏览器中它可被用来将文档转换为 XHTML。

JavaScript 解决方案

在前面的章节，我们已向您讲解如何使用 XSLT 将某个 XML 文档转换为 XHTML。我们是通过以下途径完成这个工作的：向 XML 文件添加 XSL 样式表，并通过浏览器完成转换。

即使这种方法的效果很好，在 XML 文件中包含样式表引用也不总是令人满意的（例如，在无法识别 XSLT 的浏览器这种方法就无法奏效）。

更通用的方法是使用 JavaScript 来完成转换。

通过使用 JavaScript，我们可以：

- 进行浏览器确认测试
- 根据浏览器和用户需求来使用不同的样式表

这就是 XSLT 的魅力所在！XSLT 的设计目的之一就是使数据从一种格式转换到另一种格式成为可能，同时支持不同类型的浏览器以及不同的用户需求。

客户端的 XSLT 转换一定会成为未来浏览器所执行的主要任务之一，同时我们也会看到其在特定的浏览器市场的增长（盲文、听觉浏览器、网络打印机，手持设备，等等）。

XML 文件和 XSL 文件

请看这个在前面的章节已展示过的 XML 文档：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
.
.
</catalog>
```

查看 XML 文件。

以及附随的 XSL 样式表：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th align="left">Title</th>
<th align="left">Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
<tr>
<td><xsl:value-of select="title" /></td>
<td><xsl:value-of select="artist" /></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

查看 XSL 文件。

请注意，这个 **XML** 文件没有包含对 **XSL** 文件的引用。

重要事项：上面这句话意味着，XML 文件可使用多个不同的 XSL 样式表来进行转换。

在浏览器中把 XML 转换为 XHTML

这是用于在客户端把 XML 文件转换为 XHTML 的源代码：

实例

```
<html>
<head>
<script>
function loadXMLDoc(dname)
{
if (window.ActiveXObject)
{
xhttp=new ActiveXObject("Msxml2.XMLHTTP.3.0");
}
else
{
xhttp=new XMLHttpRequest();
}
xhttp.open("GET",dname,false);
xhttp.send("");
return xhttp.responseXML;
}

function displayResult()
{
xml=loadXMLDoc("cdcatalog.xml");
xsl=loadXMLDoc("cdcatalog.xsl");
// code for IE
if (window.ActiveXObject)
{
ex=xml.transformNode(xsl);
document.getElementById("example").innerHTML=ex;
}
// code for Mozilla, Firefox, Opera, etc.
else if (document.implementation && document.implementation.createDocument)
{
xsltProcessor=new XSLTProcessor();
xsltProcessor.importStylesheet(xsl);
resultDocument = xsltProcessor.transformToFragment(xml,document);
document.getElementById("example").appendChild(resultDocument);
}
}
</script>
</head>
<body onload="displayResult()">
<div id="example" />
</body>
</html>
```

提示：假如您不了解如何编写 JavaScript，请学习我们的 JavaScript 教程。

实例解释：

loadXMLDoc() 函数

loadXMLDoc() 函数是用来加载 XML 和 XSL 文件。

它检查用户拥有的和加载文件的浏览器类型。

displayResult() 函数

该函数用来显示使用 XSL 文件定义样式的 XML 文件。

- 加载 XML 和 XSL 文件
- 测试用户拥有的浏览器类型

- 如果用户浏览器支持 ActiveX 对象：
  - 使用 transformNode() 方法把 XSL 样式表应用到 XML 文档
  - 设置当前文档（id="example"）的 body 包含已经应用样式的 XML 文档
- 如果用户的浏览器不支持 ActiveX 对象：
  - 创建一个新的 XSLTProcessor 对象并导入 XSL 文件
  - 使用 transformToFragment() 方法把 XSL 样式表应用到 XML 文档
  - 设置当前文档（id="example"）的 body 包含已经应用样式的 XML 文档

XSLT - 在服务器端

由于并非所有的浏览器都支持 XSLT，另一种解决方案是在服务器上完成 XML 至 XHTML 的转化。

跨浏览器解决方案

在前面的章节，我们讲解过如何在浏览器中使用 XSLT 来完成 XML 到 XHTML 的转化。我们创建了一段使用 XML 解析器来进行转换的 JavaScript。JavaScript 解决方案无法工作于没有 XML 解析器的浏览器。

为了让 XML 数据适用于任何类型的浏览器，我们必须在服务器上对 XML 文档进行转换，然后将其作为 XHMTL 发送回浏览器。

这是 XSLT 的另一个优点。XSLT 的设计目标之一是使数据在服务器上从一种格式转换到另一种格式成为可能，并向所有类型的浏览器返回可读的数据。

XML 文件和 XSLT 文件

请看这个在前面的章节已展示过的 XML 文档：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
.
.
</catalog>
```

查看 XML 文件。

以及附随的 XSL 样式表：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th align="left">Title</th>
<th align="left">Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
<tr>
<td><xsl:value-of select="title" /></td>
<td><xsl:value-of select="artist" /></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

查看 XSL 文件。

请注意，这个 **XML** 文件没有包含对 **XSL** 文件的引用。

重要事项：上面这句话意味着，XML 文件可使用多个不同的 XSL 样式表来进行转换。

在服务器把 XML 转换为 XHTML

这是用于在服务器上把 XML 文件转换为 XHTML 的源代码：

```
<%
'Load XML
set xml = Server.CreateObject("Microsoft.XMLDOM")
xml.async = false
xml.load(Server.MapPath("cdcatalog.xml"))

'Load XSL
set xsl = Server.CreateObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load(Server.MapPath("cdcatalog.xsl"))

'Transform file
Response.Write(xml.transformNode(xsl))
%>
```

提示：假如您不了解如何编写 ASP，您可以学习我们的 ASP 教程。

第一段代码创建了微软的 XML 解析器（XMLDOM）的一个实例，并把 XML 文件载入了内存。第二段代码创建了解析器的另一个实例，并把这个 XSL 文件载入了内存。最后一行代码使用 XSL 文档转换了 XML 文档，并把结果作为 XHTML 发送到您的浏览器。太好了！

它是如何工作的。

XSLT - 编辑 XML

存储在 XML 文件中的数据可通过因特网浏览器进行编辑。

打开、编辑并保存 XML

现在，我们会为您展示如何打开、编辑及保存存储于服务器上的 XML 文件。

我们将使用 XSL 把 XML 文档转换到一个 HTML 表单中。XML 元素的值会被写到 HTML 表单中的 HTML 输入域。这个 HTML 表单是可编辑的。在被编辑完成后，数据会被提交回服务器，XML 文件会得到更新（这部分由 ASP 完成）。

XML 文件和 XSL 文件

首先，请看将被使用的 XML 文档（"tool.xml"）：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tool>
<field id="prodName">
<value>HAMMER HG2606</value>
</field>
<field id="prodNo">
<value>32456240</value>
</field>
<field id="price">
<value>$30.00</value>
</field>
</tool>
```

查看 XML 文件。

接着，请看下面的样式表（"tool.xsl"）：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<form method="post" action="edittool.html">
<h2>Tool Information (edit):</h2>
<table border="0">
<xsl:for-each select="tool/field">
<tr>
<td><xsl:value-of select="@id"/></td>
<td>
<input type="text">
<xsl:attribute name="id">
<xsl:value-of select="@id" />
</xsl:attribute>
<xsl:attribute name="name">
<xsl:value-of select="@id" />
</xsl:attribute>
<xsl:attribute name="value">
<xsl:value-of select="value" />
</xsl:attribute>
</input>
</td>
</tr>
</xsl:for-each>
</table>
<br />
<input type="submit" id="btn_sub" name="btn_sub" value="Submit" />
<input type="reset" id="btn_res" name="btn_res" value="Reset" />
</form>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

查看 XSL 文件。

上面这个 XSL 文件会循环遍历 XML 文件中的元素，并为每个 XML "field" 元素创建一个输入域。XML "field" 元素的 "id" 属性的值被添加到每个 HTML 输入域的 "id" 和 "name" 属性。每个 XML "value" 元素的值被添加到每个 HTML 输入域的 "value" 属性。结果是，可以得到一个包含 XML 文件中值的可编辑的 HTML 表单。

然后，我们还有第二个样式表："tool_updated.xsl"。这个 XSL 文件会被用来显示已更新的 XML 数据。这个样式表不会输出可编辑 HTML 表单，而是一个静态的 HTML 表格：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>Updated Tool Information:</h2>
<table border="1">
<xsl:for-each select="tool/field">
<tr>
<td><xsl:value-of select="@id" /></td>
<td><xsl:value-of select="value" /></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

查看 XSL 文件。

ASP 文件

在上面 "tool.xsl" 文件中，HTML 表单的 action 属性的值是 "edittool.asp" 。

"edittool.asp" 页面包含两个函数：loadFile() 函数载入并转换 XML 文件，updateFile() 函数更新 XML 文件：

```
<%
function loadFile(xmlfile,xslfile)
Dim xmlDoc,xslDoc
'Load XML file
set xmlDoc = Server.CreateObject("Microsoft.XMLDOM")
xmlDoc.async = false
xmlDoc.load(xmlfile)
'Load XSL file
set xslDoc = Server.CreateObject("Microsoft.XMLDOM")
xslDoc.async = false
xslDoc.load(xslfile)
'Transform file
Response.Write(xmlDoc.transformNode(xslDoc))
end function

function updateFile(xmlfile)
Dim xmlDoc,rootEl,f
Dim i
'Load XML file
set xmlDoc = Server.CreateObject("Microsoft.XMLDOM")
xmlDoc.async = false
xmlDoc.load(xmlfile)

'Set the rootEl variable equal to the root element
Set rootEl = xmlDoc.documentElement

'Loop through the form collection
for i = 1 To Request.Form.Count
'Eliminate button elements in the form
if instr(1,Request.Form.Key(i),"btn_")=0 then
'The selectSingleNode method queries the XML file for a single node
'that matches a query. This query requests the value element that is
'the child of a field element that has an id attribute which matches
'the current key value in the Form Collection. When there is a match -
'set the text property equal to the value of the current field in the
'Form Collection.
set f = rootEl.selectSingleNode("field[@id='" & _
Request.Form.Key(i) & "']/value")
f.Text = Request.Form(i)
end if
next

'Save the modified XML file
xmlDoc.save xmlfile

'Release all object references
set xmlDoc=nothing
set rootEl=nothing
set f=nothing

'Load the modified XML file with a style sheet that
'allows the client to see the edited information
loadFile xmlfile,server.MapPath("tool_updated.xsl")
end function

'If the form has been submitted update the
'XML file and display result - if not,
'transform the XML file for editing
if Request.Form("btn_sub")="" then
loadFile server.MapPath("tool.xml"),server.MapPath("tool.xsl")
else
updateFile server.MapPath("tool.xml")
end if
%>
```

提示：假如您不了解如何编写 ASP，请学习我们的 ASP 教程。

注意：我们正在转换并更新位于服务器上的 XML 文件。这是一个跨平台的解决方案。客户端仅能获得从服务器返回的 HTML - 而 HTML 可运行于任何浏览器。

XML 编辑器

如果希望极认真地学习和使用 XML，那么您一定会从使用一款专业的 XML 编辑器中受益。

XML 是基于文本的

XML 是基于文本的标记语言。

关于 XML 的一件很重要的事情是，XML 文件可被类似记事本这样的简单的文本编辑器来创建和编辑。

不过，在您开始使用 XML 进行工作时，您很快会发现，使用一款专业的 XML 编辑器来编辑 XML 文档会更好。

为什么不使用记事本？

许多 Web 开发人员使用记事本来编辑 HTML 和 XML 文档，这是因为最常用的操作系统都带有记事本，而且它很容易使用。从个人来讲，我经常使用记事本来快速地编辑某些简单的 HTML、CSS 以及 XML 文件。

但是，如果您使用记事本对 XML 进行编辑，可能很快会发现不少问题。

记事本不能确定您编辑的文档类型，所以也就无法辅助您的工作。

为什么使用 XML 编辑器？

当今，XML 是非常重要的技术，并且开发项目正在使用这些基于 XML 的技术：

- 用 XML Schema 定义 XML 的结构和数据类型
- 用 XSLT 来转换 XML 数据
- 用 SOAP 来交换应用程序之间的 XML 数据
- 用 WSDL 来描述网络服务
- 用 RDF 来描述网络资源
- 用 XPath 和 XQuery 来访问 XML 数据
- 用 SMIL 来定义图形

为了能够编写出无错的 XML 文档，您需要一款智能的 XML 编辑器！

XML 编辑器

专业的 XML 编辑器会帮助您编写无错的 XML 文档，根据某种 DTD 或者 schema 来验证 XML，以及强制您创建合法的 XML 结构。

XML 编辑器应该具有如下能力：

- 为开始标签自动添加结束标签
- 强制您编写合法的 XML
- 根据 DTD 来验证 XML
- 根据 Schema 来验证 XML
- 对您的 XML 语法进行代码的颜色化显示

XSLT 元素参考手册

源自于 W3C 推荐标准（XSLT Version 1.0）的 XSLT 元素。

XSLT 元素

如果您需要有关下列元素的更详细的信息，请点击元素列中的链接。

元素	描述
apply-imports	应用来自导入样式表中的模版规则。
apply-templates	向当前元素或当前元素的子节点应用模板规则。
attribute	添加属性。
attribute-set	定义命名的属性集。
call-template	调用一个指定的模板。
choose	与 <when> 以及 <otherwise> 协同使用，来表达多重条件测试。
comment	在结果树中创建注释节点。
copy	创建当前节点的一个副本（无子节点及属性）。
copy-of	创建当前节点的一个副本（带有子节点及属性）。
decimal-format	定义当通过 format-number() 函数把数字转换为字符串时，所要使用的字符和符号。
element	在输出文档中创建一个元素节点。
fallback	假如处理器不支持某个 XSLT 元素，规定一段替代代码来运行。
for-each	循环遍历指定的节点集中的每个节点。
if	包含一个模板，仅当某个指定的条件成立时应用此模板。
import	用于把一个样式表中的内容导入另一个样式表中。 注意：被导入的样式表的优先级低于导出的样式表。
include	把一个样式表中的内容包含到另一个样式表中。注意： 被包含的样式表（included style sheet）拥有与包含的样式表（including style sheet）相同的优先级。
key	声明一个命名的键，该键通过 key() 函数在样式表中使用。
message	向输出写一条消息（用于报告错误）。
namespace-	把样式表中的命名空间替换为输出中不同的命名空间。

alias	把样式表中的命名空间替换为输出中不同的命名空间。
number	测定当前节点的整数位置，并对数字进行格式化。
otherwise	规定 <choose> 元素的默认动作。
output	定义输出文档的格式。
param	声明一个局部或全局参数。
preserve-space	定义保留空白的元素。
processing-instruction	向输出写一条处理指令，即生成处理指令节点。
sort	对输出进行排序。
strip-space	定义应当删除空白字符的元素。
stylesheet	定义样式表的根元素。
template	当指定的节点被匹配时所应用的规则。
text	向输出写文本，即通过样式表生成文本节点。
transform	定义样式表的根元素。
value-of	提取选定节点的值。
variable	声明局部或者全局的变量。
when	规定 <choose> 元素的动作。
with-param	规定传递给模板的参数的值。

XSLT 函数

XQuery 1.0、XPath 2.0 以及 XSLT 2.0 共享相同的函数库。

XSLT 函数

XSLT 含有超过 100 个内建的函数。这些函数用于字符串值、数值、日期和时间比较、节点和 QName 操作、序列操作、布尔值，等等。

>  函数命名空间的默认前缀是 fn。
>
>  函数命名空间的 URI 是：http://www.w3.org/2005/xpath-functions

提示：函数在被调用时常带有 fn: 前缀，比如 fn:string()。不过，既然 fn: 是命名空间的默认前缀，那么在被调用时，函数的名称不必使用前缀。

您可以在我们的 XPath 教程中访问所有内建的 XSLT 2.0 函数的参考手册。

此外，下面列出了内建的 XSLT 函数：

名称	描述
current()	返回当前节点。
document()	用于访问外部 XML 文档中的节点。
element-available()	检测 XSLT 处理器是否支持指定的元素。
format-number()	把数字转换为字符串。
function-available()	检测 XSLT 处理器是否支持指定的函数。
generate-id()	返回唯一标识指定节点的字符串值。
key()	通过使用由 <xsl:key> 元素规定的索引号返回节点集。
system-property()	返回系统属性的值。
unparsed-entity-uri()	返回未解析实体的 URI。

XPath 教程

XPath 简介

XPath 是一门在 XML 文档中查找信息的语言。

在学习之前应该具备的知识：

在您继续学习之前，应该对下面的知识有基本的了解：

- HTML / XHTML
- XML / XML Namespaces

如果您希望首先学习这些项目，请在我们的 首页 访问这些教程。

什么是 XPath?

- XPath 使用路径表达式在 XML 文档中进行导航
- XPath 包含一个标准函数库
- XPath 是 XSLT 中的主要元素
- XPath 是一个 W3C 标准

XPath 路径表达式

XPath 使用路径表达式来选取 XML 文档中的节点或者节点集。这些路径表达式和我们在常规的电脑文件系统中看到的表达式非常相似。

XPath 标准函数

XPath 含有超过 100 个内建的函数。这些函数用于字符串值、数值、日期和时间比较、节点和 QName 处理、序列处理、逻辑值等等。

XPath 在 XSLT 中使用

XPath 是 XSLT 标准中的主要元素。如果没有 XPath 方面的知识，您就无法创建 XSLT 文档。

您可以在我们的 《XSLT 教程》 中阅读更多的内容。

XQuery 和 XPointer 均构建于 XPath 表达式之上。XQuery 1.0 和 XPath 2.0 共享相同的数据模型，并支持相同的函数和运算符。

您可以在我们的《XQuery 教程》中阅读更多有关 XQuery 的知识。

XPath 是 W3C 标准

XPath 于 1999 年 11 月 16 日 成为 W3C 标准。

XPath 被设计为供 XSLT、XPointer 以及其他 XML 解析软件使用。

您可以在我们的《W3C 教程》中阅读更多有关 XPath 标准的信息。

XPath 节点

XPath 术语

节点

在 XPath 中，有七种类型的节点：元素、属性、文本、命名空间、处理指令、注释以及文档（根）节点。XML 文档是被作为节点树来对待的。树的根被称为文档节点或者根节点。

请看下面这个 XML 文档：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

上面的XML文档中的节点例子：

```
<bookstore> (文档节点)

<author>J K. Rowling</author> (元素节点)

lang="en" (属性节点)
```

基本值（或称原子值，Atomic value）

基本值是无父或无子的节点。

基本值的例子：

```
J K. Rowling

"en"
```

项目（Item）

项目是基本值或者节点。

节点关系

父（Parent）

每个元素以及属性都有一个父。

在下面的例子中，book 元素是 title、author、year 以及 price 元素的父：

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

子（Children）

元素节点可有零个、一个或多个子。

在下面的例子中，title、author、year 以及 price 元素都是 book 元素的子：

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

同胞（Sibling）

拥有相同的父的节点

在下面的例子中，title、author、year 以及 price 元素都是同胞：

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

先辈（Ancestor）

某节点的父、父的父，等等。

在下面的例子中，title 元素的先辈是 book 元素和 bookstore 元素：

```
<bookstore>

<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

后代（Descendant）

某个节点的子，子的子，等等。

在下面的例子中，bookstore 的后代是 book、title、author、year 以及 price 元素：

```
<bookstore>

<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

XPath 语法

XPath 使用路径表达式来选取 XML 文档中的节点或节点集。节点是通过沿着路径 (path) 或者步 (steps) 来选取的。

XML 实例文档

我们将在下面的例子中使用这个 XML 文档。

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

选取节点

XPath 使用路径表达式在 XML 文档中选取节点。节点是通过沿着路径或者 step 来选取的。下面列出了最有用的路径表达式：

表达式	描述
nodename	选取此节点的所有子节点。
/	从根节点选取。
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。
.	选取当前节点。
..	选取当前节点的父节点。
@	选取属性。

在下面的表格中，我们已列出了一些路径表达式以及表达式的结果：

路径表达式	结果
bookstore	选取 bookstore 元素的所有子节点。
/bookstore	选取根元素 bookstore。注释：假如路径起始于正斜杠(/)，则此路径始终代表到某元素的绝对路径！
bookstore/book	选取属于 bookstore 的子元素的所有 book 元素。
//book	选取所有 book 子元素，而不管它们在文档中的位置。
bookstore//book	选择属于 bookstore 元素的后代的所有 book 元素，而不管它们位于 bookstore 之下的什么位置。
//@lang	选取名为 lang 的所有属性。

谓语（Predicates）

谓语用来查找某个特定的节点或者包含某个指定的值的节点。

谓语被嵌在方括号中。

在下面的表格中，我们列出了带有谓语的一些路径表达式，以及表达式的结果：

路径表达式	结果
/bookstore/book[1]	选取属于 bookstore 子元素的第一个 book 元素。
/bookstore/book[last()]	选取属于 bookstore 子元素的最后一个 book 元素。
/bookstore/book[last()-1]	选取属于 bookstore 子元素的倒数第二个 book 元素。
/bookstore/book[position()<3]	选取最前面的两个属于 bookstore 元素的子元素的 book 元素。
//title[@lang]	选取所有拥有名为 lang 的属性的 title 元素。
//title[@lang='eng']	选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。
/bookstore/book[price>35.00]	选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。
/bookstore/book[price>35.00]/title	选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00。

选取未知节点

XPath 通配符可用来选取未知的 XML 元素。

通配符	描述
*	匹配任何元素节点。
@*	匹配任何属性节点。
node()	匹配任何类型的节点。

在下面的表格中，我们列出了一些路径表达式，以及这些表达式的结果：

路径表达式	结果
/bookstore/*	选取 bookstore 元素的所有子元素。
//*	选取文档中的所有元素。
//title[@*]	选取所有带有属性的 title 元素。

选取若干路径

通过在路径表达式中使用"|"运算符，您可以选取若干个路径。

在下面的表格中，我们列出了一些路径表达式，以及这些表达式的结果：

路径表达式	结果
//book/title | //book/price	选取 book 元素的所有 title 和 price 元素。
//title | //price	选取文档中的所有 title 和 price 元素。
/bookstore/book/title | //price	选取属于 bookstore 元素的 book 元素的所有 title 元素，以及文档中所有的 price 元素。

XPath 轴（Axes）

XML 实例文档

我们将在下面的例子中使用此 XML 文档：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

XPath 轴（Axes）

轴可定义相对于当前节点的节点集。

轴名称	结果
ancestor	选取当前节点的所有先辈（父、祖父等）。
ancestor-or-self	选取当前节点的所有先辈（父、祖父等）以及当前节点本身。
attribute	选取当前节点的所有属性。
child	选取当前节点的所有子元素。
descendant	选取当前节点的所有后代元素（子、孙等）。
descendant-or-self	选取当前节点的所有后代元素（子、孙等）以及当前节点本身。
following	选取文档中当前节点的结束标签之后的所有节点。
namespace	选取当前节点的所有命名空间节点。
parent	选取当前节点的父节点。
preceding	选取文档中当前节点的开始标签之前的所有节点。
preceding-sibling	选取当前节点之前的所有同级节点。
self	选取当前节点。

XPath 运算符

XPath 表达式可返回节点集、字符串、逻辑值以及数字。

XPath 运算符

下面列出了可用在 XPath 表达式中的运算符：

运算符	描述	实例	返回值
|	计算两个节点集	//book | //cd	返回所有拥有 book 和 cd 元素的节点集
+	加法	6 + 4	10
-	减法	6 - 4	2
*	乘法	6 * 4	24
div	除法	8 div 4	2
=	等于	price=9.80	如果 price 是 9.80，则返回 true。如果 price 是 9.90，则返回 false。
!=	不等于	price!=9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。
<	小于	price<9.80	如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。
<=	小于或等于	price<=9.80	如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。
>	大于	price>9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。
>=	大于或等于	price>=9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.70，则返回 false。
or	或	price=9.80 or price=9.70	如果 price 是 9.80，则返回 true。如果 price 是 9.50，则返回 false。
and	与	price>9.00 and price<9.90	如果 price 是 9.80，则返回 true。如果 price 是 8.50，则返回 false。
mod	计算除法的余数	5 mod 2	1

XPath Examples

在本节，让我们通过实例来学习一些基础的 XPath 语法。

XML实例文档

我们将在下面的例子中使用这个 XML 文档：

"books.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

在您的浏览器中查看此 "books.xml" 文件。

加载 XML 文档

所有现代浏览器都支持使用 XMLHttpRequest 来加载 XML 文档的方法。

针对大多数现代浏览器的代码：

```
var xmlhttp=new XMLHttpRequest()
```

针对古老的微软浏览器（IE 5 和 6）的代码：

```
var xmlhttp=new ActiveXObject("Microsoft.XMLHTTP")
```

选取节点

不幸的是，Internet Explorer 和其他处理 XPath 的方式不同。

在我们的例子中，包含适用于大多数主流浏览器的代码。

Internet Explorer 使用 selectNodes() 方法从 XML 文档中的选取节点：

```
xmlDoc.selectNodes(_xpath_);
```

Firefox、Chrome、Opera 以及 Safari 使用 evaluate() 方法从 XML 文档中选取节点：

```
xmlDoc.evaluate(_xpath_, xmlDoc, null, XPathResult.ANY_TYPE,null);
```

选取所有 title

下面的例子选取所有 title 节点：

实例

```
/bookstore/book/title
```

选取第一个 book 的 title

下面的例子选取 bookstore 元素下面的第一个 book 节点的 title：

实例

```
/bookstore/book[1]/title
```

这里有一个问题。上面的例子在 IE 和其他浏览器中输出不同的结果。

IE5 以及更高版本将 [0] 视为第一个节点，而根据 W3C 的标准，应该是 [1]。

一种解决方法！

为了解决 IE5+ 中 [0] 和 [1] 的问题，可以为 XPath 设置语言选择（SelectionLanguage）。

下面的例子选取 bookstore 元素下面的第一个 book 节点的 title：

实例

```
_xml_.setProperty("SelectionLanguage","XPath");
_xml_.selectNodes("/bookstore/book[1]/title");
```

选取所有价格

下面的例子选取 price 节点中的所有文本：

实例

```
/bookstore/book/price/text()
```

选取价格高于 35 的 price 节点

下面的例子选取价格高于 35 的所有 price 节点：

实例

```
/bookstore/book[price>35]/price
```

选取价格高于 35 的 title 节点

下面的例子选取价格高于 35 的所有 title 节点：

实例

```
/bookstore/book[price>35]/title
```

XPath、XQuery 以及 XSLT 函数函数参考手册

下面的参考手册定义了XPath 2.0，XQuery 1.0和XSLT 2.0中的函数。

函数参考手册

存取	错误和跟踪	数值 字符串
AnyURI	逻辑	持续时间/日期/时间
QName	节点	序列
Context		

 函数命名空间的默认前缀为 fn:

 函数命名空间的 URI为：http://www.w3.org/2005/xpath-functions

存取函数

名称	说明
fn:node-name(node)	返回参数节点的节点名称。
fn:nilled(node)	返回是否拒绝参数节点的布尔值。
fn:data(item.item,...)	接受项目序列，并返回原子值序列。
fn:base-uri() fn:base-uri(node)	返回当前节点或指定节点的 base-uri 属性的值。
fn:document-uri(node)	返回指定节点的 document-uri 属性的值。

错误和跟踪函数

名称	说明
fn:error() fn:error(error) fn:error(error,description) fn:error(error,description,error-object)	例子：error(fn:QName('http://example.com/test', 'err:toohigh'), 'Error: Price is too high') 结果：向外部处理环境返回 http://example.com/test#toohigh 以及字符串 "Error: Price is too high"。
fn:trace(value,label)	用于对查询进行 debug。

有关数值的函数

名称	说明
fn:number(arg)	返回参数的数值。参数可以是布尔值、字符串或节点集。例子：number('100') 结果：100
fn:abs(num)	返回参数的绝对值。例子：abs(3.14) 结果：3.14 例子：abs(-3.14) 结果：3.14
fn:ceiling(num)	返回大于 num 参数的最小整数。例子：ceiling(3.14) 结果：4
fn:floor(num)	返回不大于 num 参数的最大整数。例子：floor(3.14) 结果：3
fn:round(num)	把 num 参数舍入为最接近的整数。例子：round(3.14) 结果：3
fn:round-half-to-even()	例子：round-half-to-even(0.5) 结果：0 例子：round-half-to-even(1.5) 结果：2 例子：round-half-to-even(2.5) 结果：2

有关字符串的函数

名称	说明
fn:string(arg)	返回参数的字符串值。参数可以是数字、逻辑值或节点集。 例子：string(314) 结果："314"
fn:codepoints-to-string(int,int,...)	根据代码点序列返回字符串。例子：codepoints-to-string(84, 104, 233, 114, 232, 115, 101) 结果：'Thérèse'
fn:string-to-codepoints(string)	根据字符串返回代码点序列。例子：string-to-codepoints("Thérèse") 结果：84, 104, 233, 114, 232, 115, 101
fn:codepoint-equal(comp1,comp2)	根据 Unicode 代码点对照，如果 comp1 的值等于 comp2 的值，则返回 true。(http://www.w3.org/2005/02/xpath-functions/collation/codepoint)，否则返回 false。
fn:compare(comp1,comp2) fn:compare(comp1,comp2,collation)	如果 comp1 小于 comp2，则返回 -1。如果 comp1 等于 comp2，则返回 0。如果 comp1 大于 comp2，则返回 1。（根据所用的对照规则）。例子：compare('ghi', 'ghi') 结果：0
fn:concat(string,string,...)	返回字符串的拼接。例子：concat('XPath ','is ','FUN!') 结果：'XPath is FUN!'
fn:string-join((string,string,...),sep)	使用 sep 参数作为分隔符，来返回 string 参数拼接后的字符串。例子：string-join(('We', 'are', 'having', 'fun!'), ' ') 结果：' We are having fun! ' 例子：string-join(('We', 'are', 'having', 'fun!')) 结果：'Wearehavingfun!' 例子：string-join((), 'sep') 结果：''
fn:substring(string,start,len) fn:substring(string,start)	返回从 start 位置开始的指定长度的子字符串。第一个字符的下标是 1。如果省略 len 参数，则返回从位置 start 到字符串末尾的子字符串。例子：

fn:substring(string,start)	substring('Beatles',1,4) 结果：'Beat' 例子：substring('Beatles',2) 结果：'eatles'
fn:string-length(string) fn:string-length()	返回指定字符串的长度。如果没有 string 参数，则返回当前节点的字符串值的长度。例子：string-length('Beatles') 结果：7
fn:normalize-space(string) fn:normalize-space()	删除指定字符串的开头和结尾的空白，并把内部的所有空白序列替换为一个，然后返回结果。如果没有 string 参数，则处理当前节点。例子：normalize-space(' The XML ') 结果：'The XML'
fn:normalize-unicode()	执行 Unicode 规格化。
fn:upper-case(string)	把 string 参数转换为大写。例子：upper-case('The XML') 结果：'THE XML'
fn:lower-case(string)	把 string 参数转换为小写。例子：lower-case('The XML') 结果：'the xml'
fn:translate(string1,string2,string3)	把 string1 中的 string2 替换为 string3。例子：translate('12:30','30','45') 结果：'12:45' 例子：translate('12:30','03','54') 结果：'12:45' 例子：translate('12:30','0123','abcd') 结果：'bc:da'
fn:escape-uri(stringURI,esc-res)	例子：escape-uri("http://example.com/test#car", true()) 结果："http%3A%2F%2Fexample.com%2Ftest#car" 例子：escape-uri("http://example.com/test#car", false()) 结果："http://example.com/test#car" 例子：escape-uri ("http://example.com/~bébé", false()) 结果："http://example.com/~b%C3%A9b%C3%A9"
fn:contains(string1,string2)	如果 string1 包含 string2，则返回 true，否则返回 false。例子：contains('XML','XM') 结果：true
fn:starts-with(string1,string2)	如果 string1 以 string2 开始，则返回 true，否则返回 false。例子：starts-with('XML','X') 结果：true
fn:ends-with(string1,string2)	如果 string1 以 string2 结尾，则返回 true，否则返回 false。例子：ends-with('XML','X') 结果：false
fn:substring-before(string1,string2)	返回 string2 在 string1 中出现之前的子字符串。例子：substring-before('12/10','/') 结果：'12'
fn:substring-after(string1,string2)	返回 string2 在 string1 中出现之后的子字符串。例子：substring-after('12/10','/') 结果：'10'
fn:matches(string,pattern)	如果 string 参数匹配指定的模式，则返回 true，否则返回 false。例子：matches("Merano", "ran") 结果：true
fn:replace(string,pattern,replace)	把指定的模式替换为 replace 参数，并返回结果。例子：replace("Bella Italia", "l", "") 结果：'Be**a Itaia' 例子：replace("Bella Italia", "l", "") 结

fn:tokenize(string,pattern)	例子：tokenize("XPath is fun", "\s+") 结果：("XPath", "is", "fun")

针对 anyURI 的函数

名称	说明
fn:resolve-uri(relative,base)	

关于布尔值的函数

名称	说明
fn:boolean(arg)	返回数字、字符串或节点集的布尔值。
fn:not(arg)	首先通过 boolean() 函数把参数还原为一个布尔值。如果该布尔值为 false，则返回 true，否则返回 true。例子：not(true()) 结果：false
fn:true()	返回布尔值 true。例子：true() 结果：true
fn:false()	返回布尔值 false。例子：false() 结果：false

有关持续时间、日期和时间的函数

日期、时间、持续时间的组件提取函数

名称	说明
fn:dateTime(date,time)	把参数转换为日期和时间。
fn:years-from-duration(datetimedur)	返回参数值的年份部分的整数，以标准词汇表示法来表示。
fn:months-from-duration(datetimedur)	返回参数值的月份部分的整数，以标准词汇表示法来表示。
fn:days-from-duration(datetimedur)	返回参数值的天部分的整数，以标准词汇表示法来表示。
fn:hours-from-duration(datetimedur)	返回参数值的小时部分的整数，以标准词汇表示法来表示。
fn:minutes-from-duration(datetimedur)	返回参数值的分钟部分的整数，以标准词汇表示法来表示。
fn:seconds-from-duration(datetimedur)	返回参数值的分钟部分的十进制数，以标准词汇表示法来表示。
fn:year-from-dateTime(datetime)	返回参数本地值的年部分的整数。例子：year-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10")) 结果：2005

fn:month-from-dateTime(datetime)	返回参数本地值的月部分的整数。例子：month-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10")) 结果：01
fn:day-from-dateTime(datetime)	返回参数本地值的天部分的整数。例子：day-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10")) 结果：10
fn:hours-from-dateTime(datetime)	返回参数本地值的小时部分的整数。例子：hours-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10")) 结果：12
fn:minutes-from-dateTime(datetime)	返回参数本地值的分钟部分的整数。例子：minutes-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10")) 结果：30
fn:seconds-from-dateTime(datetime)	返回参数本地值的秒部分的十进制数。例子：seconds-from-dateTime(xs:dateTime("2005-01-10T12:30:00-04:10")) 结果：0
fn:timezone-from-dateTime(datetime)	返回参数的时区部分，如果存在。
fn:year-from-date(date)	返回参数本地值中表示年的整数。例子：year-from-date(xs:date("2005-04-23"))结果：2005
fn:month-from-date(date)	返回参数本地值中表示月的整数。例子：month-from-date(xs:date("2005-04-23")) 结果：4
fn:day-from-date(date)	返回参数本地值中表示天的整数。例子：day-from-date(xs:date("2005-04-23")) 结果：23
fn:timezone-from-date(date)	返回参数的时区部分，如果存在。
fn:hours-from-time(time)	返回参数本地值中表示小时部分的整数。例子：hours-from-time(xs:time("10:22:00")) 结果：10
fn:minutes-from-time(time)	返回参数本地值中表示分钟部分的整数。例子：minutes-from-time(xs:time("10:22:00")) 结果：22
fn:seconds-from-time(time)	返回参数本地值中表示秒部分的整数。例子：seconds-from-time(xs:time("10:22:00")) 结果：0
fn:timezone-from-time(time)	返回参数的时区部分，如果存在。
fn:adjust-dateTime-to-timezone(datetime,timezone)	如果 timezone 参数为空，则返回没有时区的 dateTime。否则返回带有时区的 dateTime。
fn:adjust-date-to-timezone(date,timezone)	如果 timezone 参数为空，则返回没有时区的 date。否则返回带有时区的 date。
fn:adjust-time-to-timezone(time,timezone)	如果 timezone 参数为空，则返回没有时区的 time。否则返回带有时区的 time。

与 QNames 相关的函数

名称	说明
fn:QName()	
fn:local-name-from-QName()	
fn:namespace-uri-from-QName()	
fn:namespace-uri-for-prefix()	
fn:in-scope-prefixes()	
fn:resolve-QName()	

关于节点的函数

名称	说明
fn:name() fn:name(nodeset)	返回当前节点的名称或指定节点集中的第一个节点。
fn:local-name() fn:local-name(nodeset)	返回当前节点的名称或指定节点集中的第一个节点 - 不带有命名空间前缀。
fn:namespace-uri() fn:namespace-uri(nodeset)	返回当前节点或指定节点集中第一个节点的命名空间 URI。
fn:lang(lang)	如果当前节点的语言匹配指定的语言，则返回 true。例子：Lang("en") is true for <p xml:lang="en">...</p>例子：Lang("de") is false for <p xml:lang="en">...</p>
fn:root() fn:root(node)	返回当前节点或指定的节点所属的节点树的根节点。通常是文档节点。

有关序列的函数

一般性的函数

名称	说明
fn:index-of((item,item,...),searchitem)	返回在项目序列中等于 searchitem 参数的位置。例子：index-of ((15, 40, 25, 40, 10), 40) 结果：(2, 4) 例子：index-of (("a", "dog", "and", "a", "duck"), "a") Result (1, 4) 例子：index-of ((15, 40, 25, 40, 10), 18) 结果：()
fn:remove((item,item,...),position)	返回由 item 参数构造的新序列 - 同时删除 position 参数指定的项目。例子：remove(("ab", "cd", "ef"), 0) 结果：("ab", "cd", "ef") 例子：remove(("ab", "cd", "ef"), 1) 结果：("cd", "ef") 例子：remove(("ab", "cd", "ef"), 4) 结果：("ab", "cd", "ef")
fn:empty(item,item,...)	如果参数值是空序列，则返回 true，否则返回 false。例子：empty(remove(("ab", "cd"), 1)) 结果：false
fn:exists(item,item,...)	如果参数值不是空序列，则返回 true，否则返回 false。 例子：exists(remove(("ab"), 1)) 结果：false
fn:distinct-values((item,item,...),collation)	返回唯一不同的值。 例子：distinct-values((1, 2, 3, 1, 2)) 结果：(1, 2, 3)
fn:insert-before((item,item,...),pos,inserts)	返回由 item 参数构造的新序列 - 同时在 pos 参数指定位置插入 inserts 参数的值。例子：insert-before(("ab", "cd"), 0, "gh") 结果：("gh", "ab", "cd") 例子：insert-before(("ab", "cd"), 1, "gh") 结果：("gh", "ab", "cd") 例子：insert-before(("ab", "cd"), 2, "gh") 结果：("ab", "gh", "cd") 例子：insert-before(("ab", "cd"), 5, "gh") 结果：("ab", "cd", "gh")
fn:reverse((item,item,...))	返回指定的项目的颠倒顺序。 例子：reverse(("ab", "cd", "ef")) 结果：("ef", "cd", "ab") 例子：reverse(("ab")) 结果：("ab")
fn:subsequence((item,item,...),start,len)	返回 start 参数指定的位置返回项目序列，序列的长度由 len 参数指定。第一个项目的位置是 1。 例子：subsequence(($item1, $item2, $item3,...), 3) 结果：($item3, ...) 例子：subsequence(($item1, $item2, $item3, ...), 2, 2) 结果：($item2, $item3)
fn:unordered((item,item,...))	依据实现决定的顺序来返回项目。

测试序列容量的函数

名称	说明
fn:zero-or-one(item,item,...)	如果参数包含零个或一个项目，则返回参数，否则生成错误。
fn:one-or-more(item,item,...)	如果参数包含一个或多个项目，则返回参数，否则生成错误。
fn:exactly-one(item,item,...)	如果参数包含一个项目，则返回参数，否则生成错误。

Equals, Union, Intersection and Except

名称	说明
fn:deep-equal(param1,param2,collation)	如果 param1 和 param2 与彼此相等（deep-equal），则返回 true，否则返回 false。

合计函数

名称	说明
fn:count((item,item,...))	返回节点的数量。
fn:avg((arg,arg,...))	返回参数值的平均数。 例子：avg((1,2,3)) 结果：2
fn:max((arg,arg,...))	返回大于其它参数的参数。 例子：max((1,2,3)) 结果：3 例子：max(('a', 'k')) 结果：'k'
fn:min((arg,arg,...))	返回小于其它参数的参数。 例子：min((1,2,3)) 结果：1 例子：min(('a', 'k')) 结果：'a'
fn:sum(arg,arg,...)	返回指定节点集中每个节点的数值的总和。

生成序列的函数

名称	说明
fn:id((string,string,...),node)	Returns a sequence of element nodes that have an ID value equal to the value of one or more of the values specified in the string argument
fn:idref((string,string,...),node)	Returns a sequence of element or attribute nodes that have an IDREF value equal to the value of one or more of the values specified in the string argument
fn:doc(URI)	
fn:doc-available(URI)	如果 doc() 函数返回文档节点，则返回 true，否则返回 false。
fn:collection() fn:collection(string)	

上下文函数

名称	说明
fn:position()	返回当前正在被处理的节点的 index 位置。例子：//book[position()<=3] 结果：选择前三个 book 元素
fn:last()	返回在被处理的节点列表中的项目数目。例子：//book[last()] 结果：选择最后一个 book 元素
fn:current-dateTime()	返回当前的 dateTime（带有时区）。
fn:current-date()	返回当前的日期（带有时区）。
fn:current-time()	返回当前的时间（带有时区）。
fn:implicit-timezone()	返回隐式时区的值。
fn:default-collation()	返回默认对照的值。
fn:static-base-uri()	返回 base-uri 的值。

XQuery 教程

XQuery 简介

解释 XQuery 最佳方式是这样讲：XQuery 相对于 XML 的关系，等同于 SQL 相对于数据库表的关系。

XQuery 被设计用来查询 XML 数据 - 不仅仅限于 XML 文件，还包括任何可以 XML 形态呈现的数据，包括数据库。

您应该具备的基础知识：

在您继续学习之前，需要对下面的知识有基本的了解：

- HTML / XHTML
- XML / XML 命名空间
- XPath

如果您希望首先学习这些项目，请在我们的 首页 访问这些教程。

什么是 XQuery?

- XQuery 是用于 XML 数据查询的语言
- XQuery 对 XML 的作用类似 SQL 对数据库的作用
- XQuery 建立在 XPath 表达式之上
- XQuery 被所有主要的数据库引擎支持（IBM、Oracle、Microsoft等等）
- XQuery 是 W3C 标准

XQuery 和 XML 查询有关

XQuery 是用来从 XML 文档查找和提取元素及属性的语言。

这是一个 XQuery 解决实际问题的例子：

"从存储在名为 cd_catalog.xml 的 XML 文档中的 CD 集那里选取所有价格低于 10 美元的 CD 记录。"

XQuery 与 XPath

XQuery 1.0 和 XPath 2.0 共享相同的数据模型，并支持相同的函数和运算符。假如您已经学习了 XPath，那么学习 XQuery 也不会有问题。

您可以在我们的《XPath 教程》中阅读更多有关 XPath 的知识。

XQuery - 应用举例

XQuery 可用于：

- 提取信息以便在网络服务中使用
- 生成摘要报告
- 把 XML 数据转换为 XHTML
- 为获得相关信息而搜索网络文档

XQuery 是一个 W3C 推荐标准

XQuery 与多种 W3C 标准相兼容，比如 XML、Namespaces、XSLT、XPath 以及 XML Schema。

XQuery 1.0 在 2007年1月23日 被确立为 W3C 推荐标准。

如需获得更多有关 W3C 的 XQuery 活动的信息，请阅读我们的《W3C 教程》。

XQuery 实例

在本节，让我们通过研究一个例子来学习一些基础的 XQuery 语法。

XML 实例文档

我们将在下面的例子中使用这个 XML 文档。

"books.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

在您的浏览器中查看 "books.xml" 文件。

如何从 "books.xml" 选取节点？

函数

XQuery 使用函数来提取 XML 文档中的数据。

doc() 用于打开 "books.xml" 文件：

```
doc("books.xml")
```

路径表达式

XQuery 使用路径表达式在 XML 文档中通过元素进行导航。

下面的路径表达式用于在 "books.xml" 文件中选取所有的 title 元素：

```
doc("books.xml")/bookstore/book/title
```

(/bookstore 选取 bookstore 元素，/book 选取 bookstore 元素下的所有 book 元素，而 /title 选取每个 book 元素下的所有 title 元素)

上面的 XQuery 可提取以下数据：

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

谓语

XQuery 使用谓语来限定从 XML 文档所提取的数据。

下面的谓语用于选取 bookstore 元素下的所有 book 元素，并且所选取的 book 元素下的 price 元素的值必须小于 30：

```
doc("books.xml")/bookstore/book[price<30]
```

上面的 XQuery 可提取到下面的数据：

```
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

XQuery FLWOR 表达式

XML 实例文档

我们将在下面的例子中继续使用这个 "books.xml" 文档（与上一节中的 XML 文件相同）。

在您的浏览器中查看 "books.xml" 文件。

如果使用 FLWOR 从 "books.xml" 选取节点

请看下面这个路径表达式：

```
doc("books.xml")/bookstore/book[price>30]/title
```

上面这个表达式可选取 bookstore 元素下的 book 元素下所有的 title 元素，并且其中的 price 元素的值必须大于 30。

下面这个 FLWOR 表达式所选取的数据和上面的路径表达式是相同的：

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

输出结果：

```
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

通过 FLWOR，您可以对结果进行排序：

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

FLWOR 是 "For, Let, Where, Order by, Return" 的只取首字母缩写。

for 语句把 bookstore 元素下的所有 book 元素提取到名为 $x 的变量中。

where 语句选取了 price 元素值大于 30 的 book 元素。

order by 语句定义了排序次序。将根据 title 元素进行排序。

return 语句规定返回什么内容。在此返回的是 title 元素。

上面的 XQuery 表达式的结果：

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

XQuery FLWOR + HTML

XML 实例文档

我们将在下面的例子中继续使用这个 "books.xml" 文档（与上一节中的文件相同）。

在您的浏览器中查看 "books.xml" 文件。

在一个 HTML 列表中提交结果

请看下面的 XQuery FLWOR 表达式：

```
for $x in doc("books.xml")/bookstore/book/title
order by $x
return $x
```

上面的表达式会选取 bookstore 元素下的 book 元素下的所有 title 元素，并以字母顺序返回 title 元素。

现在，我们希望使用 HTML 列表列出我们的书店中所有的书目。我们向 FLWOR 表达式添加 <ul> 和 <li> 标签：

```
<ul>
{
for $x in doc("books.xml")/bookstore/book/title
order by $x
return <li>{$x}</li>
}
</ul>
```

以上代码输出结果：

```
<ul>
<li><title lang="en">Everyday Italian</title></li>
<li><title lang="en">Harry Potter</title></li>
<li><title lang="en">Learning XML</title></li>
<li><title lang="en">XQuery Kick Start</title></li>
</ul>
```

现在我们希望去除 title 元素，而仅仅显示 title 元素内的数据。

```
<ul>
{
for $x in doc("books.xml")/bookstore/book/title
order by $x
return <li>{data($x)}</li>
}
</ul>
```

结果将是一个 HTML 列表：

```
<ul>
<li>Everyday Italian</li>
<li>Harry Potter</li>
<li>Learning XML</li>
<li>XQuery Kick Start</li>
</ul>
```

XQuery 术语

在 XQuery 中，有七种节点：元素、属性、文本、命名空间、处理指令、注释、以及文档节点（或称为根节点）。

XQuery 术语

节点

在 XQuery 中，有七种节点：元素、属性、文本、命名空间、处理指令、注释、以及文档（根）节点。XML 文档是被作为节点树来对待的。树的根被称为文档节点或者根节点。

请看下面的 XML 文档：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book>
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

上面的 XML 文档中的节点例子：

```
<bookstore> (文档节点)

<author>J K. Rowling</author> (元素节点)

lang="en" (属性节点)
```

基本值是无父或无子的节点。

基本值的例子：

```
J K. Rowling

"en"
```

项目

项目是基本值或者节点。

节点关系

父（Parent）

每个元素以及属性都有一个父。

在下面的例子中，book 元素是 title、author、year 以及 price 元素的父：

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

子（Children）

节点元素可有零个、一个或多个子。

在下面的例子中，title、author、year 以及 price 元素都是 book 元素的子：

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

同胞（Sibling）

拥有相同的父的节点。

在下面的例子中，title、author、year 以及 price 元素都是同胞：

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

先辈（Ancestor）

某节点的父、父的父，等等。

在下面的例子中，title 元素的先辈是 book 元素和 bookstore元素：

```
<bookstore>

<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

后代（Descendant）

某个节点的子，子的子，等等。

在下面的例子中，bookstore 的后代是 book、title、author、year 以及 price元素：

```
<bookstore>

<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

XQuery 语法

XQuery 对大小写敏感，XQuery 的元素、属性以及变量必须是合法的 XML 名称。

XQuery 的基础语法规则：

一些基本的语法规则：

- XQuery 对大小写敏感
- XQuery 的元素、属性以及变量必须是合法的 XML 名称。
- XQuery 字符串值可使用单引号或双引号。
- XQuery 变量由 "$" 并跟随一个名称来进行定义，举例，$bookstore
- XQuery 注释被 (: 和 :) 分割，例如，(: XQuery 注释 :)

XQuery 条件表达式

"If-Then-Else" 可以在 XQuery 中使用。

请看下面的例子：

```
for $x in doc("books.xml")/bookstore/book
return if ($x/@category="CHILDREN")
then <child>{data($x/title)}</child>
else <adult>{data($x/title)}</adult>
```

请注意 "If-Then-Else" 的语法：if 表达式后的圆括号是必需的。else 也是必需的，不过只写 "else ()" 也可以。

上面的例子的结果：

```
<adult>Everyday Italian</adult>
<child>Harry Potter</child>
<adult>Learning XML</adult>
<adult>XQuery Kick Start</adult>
```

XQuery 比较

在 XQuery 中，有两种方法来比较值。

1. 通用比较：=, !=, <, <=, >, >=
2. 值的比较：eq、ne、lt、le、gt、ge

这两种比较方法的差异如下：

请看下面的 XQuery 表达式：

```
$bookstore//book/@q > 10
```

如果 q 属性的值大于 10，上面的表达式的返回值为 true。

如下实例，如果仅返回一个 q，且它的值大于 10，那么表达式返回 true。如果不止一个 q 被返回，则会发生错误：

```
$bookstore//book/@q gt 10
```

XQuery 添加元素 和属性

XML 实例文档

我们将在下面的例子中继续使用这个 "books.xml" 文档（和上面的章节所使用的 XML 文件相同）。

在您的浏览器中查看 "books.xml" 文件。

向结果添加元素和属性

正如在前面一节看到的，我们可以在结果中引用输入文件中的元素和属性：

```
for $x in doc("books.xml")/bookstore/book/title
order by $x
return $x
```

上面的 XQuery 表达式会在结果中引用 title 元素和 lang 属性，就像这样：

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

以上 XQuery 表达式返回 title 元素的方式和它们在输入文档中被描述的方式的相同的。

现在我们要向结果添加我们自己的元素和属性！

添加 HTML 元素和文本

现在，我们要向结果添加 HTML 元素。我们会把结果放在一个 HTML 列表中：

```
<html>
<body>

<h1>Bookstore</h1>

<ul>
{
for $x in doc("books.xml")/bookstore/book
order by $x/title
return <li>{data($x/title)}. Category: {data($x/@category)}</li>
}
</ul>

</body>
</html>
```

以上 XQuery 表达式会生成下面的结果：

```
<html>
<body>

<h1>Bookstore</h1>

<ul>
<li>Everyday Italian. Category: COOKING</li>
<li>Harry Potter. Category: CHILDREN</li>
<li>Learning XML. Category: WEB</li>
<li>XQuery Kick Start. Category: WEB</li>
</ul>

</body>
</html>
```

向 HTML 元素添加属性

接下来，我们要把 category 属性作为 HTML 列表中的 class 属性来使用：

```
<html>
<body>

<h1>Bookstore</h1>

<ul>
{
for $x in doc("books.xml")/bookstore/book
order by $x/title
return <li class="{data($x/@category)}">{data($x/title)}</li>
}
</ul>

</body>
</html>
```

上面的 XQuery 表达式可生成以下结果：

```
<html>
<body>
<h1>Bookstore</h1>

<ul>
<li class="COOKING">Everyday Italian</li>
<li class="CHILDREN">Harry Potter</li>
<li class="WEB">Learning XML</li>
<li class="WEB">XQuery Kick Start</li>
</ul>

</body>
</html>
```

XQuery 选择 和 过滤

XML实例文档

我们将在下面的例子中继续使用这个 "books.xml" 文档（和上面的章节所使用的 XML 文件相同）。

在您的浏览器中查看 "books.xml" 文件。

选择和过滤元素

正如在前面的章节所看到的，我们使用路径表达式或 FLWOR 表达式来选取和过滤元素。

请看下面的 FLWOR 表达式：

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

- for - （可选） 向每个由 in 表达式返回的项目捆绑一个变量
- let - （可选）
- where - （可选） 设定一个条件
- order by - （可选） 设定结果的排列顺序
- return - 规定在结果中返回的内容

for 语句

for 语句可将变量捆绑到由 in 表达式返回的每个项目。for 语句可产生迭代。在同一个 FLWOR 表达式中可存在多重 for 语句。

如需在一个 for 语句中进行指定次数地循环，您可使用关键词 to ：

```
for $x in (1 to 5)
return <test>{$x}</test>
```

结果：

```
<test>1</test>
<test>2</test>
<test>3</test>
<test>4</test>
<test>5</test>
```

关键词 **at** 可用于计算迭代：

```
for $x at $i in doc("books.xml")/bookstore/book/title
return <book>{$i}. {data($x)}</book>
```

结果：

```
<book>1\. Everyday Italian</book>
<book>2\. Harry Potter</book>
<book>3\. XQuery Kick Start</book>
<book>4\. Learning XML</book>
```

在 for 语句中同样允许多个 in 表达式。请使用逗号来分割每一个 in 表达式：

```
for $x in (10,20), $y in (100,200)
return <test>x={$x} and y={$y}</test>
```

结果：

```
<test>x=10 and y=100</test>
<test>x=10 and y=200</test>
<test>x=20 and y=100</test>
<test>x=20 and y=200</test>
```

let 语句

let 语句可完成变量分配，并可避免多次重复相同的表达式。let 语句不会导致迭代。

```
let $x := (1 to 5)
return <test>{$x}</test>
```

结果：

```
<test>1 2 3 4 5</test>
```

where 语句

where 语句用于为结果设定一个或多个条件（criteria）。

```
where $x/price>30 and $x/price<100
```

order by 语句

order by 语句用于规定结果的排序次序。在这里，我们要根据 category 和 title 来对结果进行排序：

```
for $x in doc("books.xml")/bookstore/book
order by $x/@category, $x/title
return $x/title
```

结果：

```
<title lang="en">Harry Potter</title>
<title lang="en">Everyday Italian</title>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

return 语句：

return 语句规定要返回的内容。

```
for $x in doc("books.xml")/bookstore/book
return $x/title
```

结果：

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

XQuery 函数

XQuery 1.0、XPath 2.0 以及 XSLT 2.0 共享相同的函数库。

XQuery 函数

XQuery 含有超过 100 个内建的函数。这些函数可用于字符串值、数值、日期以及时间比较、节点和 QName 操作、序列操作、逻辑值等等。您也可在 XQuery 中定义自己的函数。

XQuery 内建函数

XQuery 函数命名空间的 URI：

http://www.w3.org/2005/02/xpath-functions

函数命名空间的默认前缀是 fn:。

提示：函数经常被通过 fn: 前缀进行调用，例如 fn:string()。不过，由于 fn: 是命名空间的默认前缀，所以函数名称不必在被调用时使用前缀。

您可以在我们的 XPath 教程中找到完整的《内建 XQuery 函数参考手册》。

函数调用实例

函数调用可与表达式一同使用。请看下面的例子：

例1：在元素中

```
<name>{upper-case($booktitle)}</name>
```

例2: 在路径表达式的谓语中

```
doc("books.xml")/bookstore/book[substring(title,1,5)='Harry']
```

例3: 在 let 语句中

```
let $name := (substring($booktitle,1,4))
```

XQuery 用户定义函数

如果找不到所需的 XQuery 函数，你可以编写自己的函数。

可在查询中或独立的库中定义用户自定义函数。

语法

```
declare function _前缀:函数名_($_参数_ AS _数据类型_)
AS _返回的数据类型_
{
 ..._函数代码_...
}
```

关于用户自定义函数的注意事项：

- 请使用 declare function 关键词
- 函数名须使用前缀
- 参数的数据类型通常与在 XML Schema 中定义的数据类型一致
- 函数主体须被花括号包围

一个在查询中声明的用户自定义函数的例子：

```
declare function local:minPrice($p as xs:decimal?,$d as xs:decimal?)
AS xs:decimal?
{
let $disc := ($p * $d) div 100
return ($p - $disc)
}

Below is an example of how to call the function above:

<minPrice>{local:minPrice($book/price,$book/discount)}</minPrice>
```

XQuery 参考手册

XQuery 1.0 和 XPath 2.0 分享相同的数据模型，并支持相同的函数和运算符。

XQuery 函数

XQuery 构建在 XPath 表达式之上。XQuery 1.0 和 XPath 2.0 分享相同的数据模型，并支持相同的函数和运算符。

XPath Operators

XPath Functions

XQuery 数据类型

XQuery 分享与 XML Schema 1.0 (XSD) 相同的数据类型。

XSD String

XSD Date

XSD Numeric

XSD Misc

XLink 和 XPointer 教程

XLink 和 XPointer 简介

XLink 定义了一套标准的在 XML 文档中创建超级链接的方法。

XPointer 使超级链接可以指向 XML 文档中更多具体的部分（片断）。

您应当具备的基础知识

学习本教程前您应当具备的基础知识:

- HTML / XHTML
- XML / XML 命名空间
- XPath

如果您希望首先学习这些项目，请在我们对 首页 访问这些教程。首页.

什么是 XLink？

- XLink 是 XML 链接语言（XML Linking Language）的缩写
- XLink 是用于在 XML 文档中创建超级链接的语言
- XLink 类似于 HTML 链接 - 但是更为强大
- XML 文档中的任何元素均可成为 XLink
- XLink 支持简易链接，也支持可将多重资源链接在一起的扩展链接
- 通过 XLink，链接可在被链接文件外进行定义
- XLink 是 W3C 推荐标准

什么是 XPointer？

- XPointer 是 XML 指针文件（XML Pointer Language）的缩写
- XPointer 使超级链接可以指向 XML 文档中更多具体的部分（片断）
- XPointer 使用 XPath 表达式在 XML 文档中进行定位
- XPointer 是 W3C 推荐标准

XLink 和 XPointer 是 W3C 标准

在2001年6月27日，XLink 被确立为 W3C 推荐标准。

XPointer 于2003年3月25日成为 W3C 推荐标准。

您可以在我们的《W3C 教程》中阅读更多有关 XML 标准的内容。

XLink 和 XPointer 的浏览器支持

浏览器只在最小限度内支持 XLink 和 XPointer。

在 Mozilla 0.98+、Netscape 6.02+ 以及 Internet Explorer 6.0 中，均具有对 XLink 某种程度的支持。更早版本的浏览器根本不支持 XLink。

XLink 和 XPointer 语法

XLink 语法

在 HTML 中，我们知道 <a> 元素可定义超级链接。不过 XML 不是这样工作的。在 XML 文档中，您可以使用任何你需要的名称 - 因此对于浏览器来说是无法预知在 XML 文档中可调用何种超级链接元素。

在 XML 文档中定义超级链接的方法是在元素上放置可用作超级链接的标记。

下面是在 XML 文档中使用 XLink 来创建链接的简单实例：

```
<?xml version="1.0"?>

<homepages xmlns:xlink="http://www.w3.org/1999/xlink">

<homepage xlink:type="simple"
xlink:href="http://www.w3schools.com">Visit W3Schools</homepage>

<homepage xlink:type="simple"
xlink:href="http://www.w3.org">Visit W3C</homepage>

</homepages>
```

为了访问 XLink 的属性和特性，我们必须在文档的顶端声明 XLink 命名空间。

XLink 的命名空间是："http://www.w3.org/1999/xlink"。

<homepage> 元素中的 xlink:type 和 xlink:href 属性定义了来自 XLink 命名空间的 type 和 href 属性。

xlink:type="simple" 可创建一个简单的两端链接（意思是"从这里到哪里"）。稍后我们会研究多端链接（多方向）。

XPointer 语法

在 HTML 中，我们可创建一个既指向某个 HTML 页面又指向 HTML 页面内某个书签的超级链接（使用#）。

有时，可指向更多具体的内容会更有好处。举例，我们需要指向某个特定的列表的第三个项目，或者指向第五段的第二行。通过 XPointer 是很容易做到的。

假如超级链接指向某个 XML 文档，我们可以在 xlink:href 属性中把 XPointer 部分添加到 URL 后面，这样就可以导航（通过 XPath 表达式）到文档中某个具体的位置了。

举例，在下面的例子中，我们通过唯一的 id "rock" 使用 XPointer 指向某个列表中的第五个项目。

```
href="http://www.example.com/cdlist.xml#id('rock').child(5,item)"
```

XLink 实例

让我们通过研究一个实例来学习一些基础的 XLink 语法。

XML 实例文档

请看下面的 XML 文档，"bookstore.xml"，它用来呈现书籍：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore xmlns:xlink="http://www.w3.org/1999/xlink">

<book title="Harry Potter">
  <description
  xlink:type="simple"
  xlink:href="http://book.com/images/HPotter.gif"
  xlink:show="new">
  As his fifth year at Hogwarts School of Witchcraft and
  Wizardry approaches, 15-year-old Harry Potter is.......
  </description>
</book>

<book title="XQuery Kick Start">
  <description
  xlink:type="simple"
  xlink:href="http://book.com/images/XQuery.gif"
  xlink:show="new">
  XQuery Kick Start delivers a concise introduction
  to the XQuery standard.......
  </description>
</book>

</bookstore>
```

在您的浏览器查看 "bookstore.xml" bookstore.xml文件。

在上面的例子中，XLink 文档命名空间(xmlns:xlink="http://www.w3.org/1999/xlink")被声明于文档的顶部。 这意味着文档可访问 XLink 的属性和特性。

xlink:type="simple" 可创建简单的类似 HTML 的链接。您也可以规定更多的复杂的链接（多方向链接），但是目前，我们仅使用简易链接。

xlink:href 属性规定了要链接的 URL，而 xlink:show 属性规定了在何处打开链接。xlink:show="new" 意味着链接（在此例中，是一幅图像）会在新窗口打开。

XLink - 深入学习

在上面的例子中，我们只展示了简单的链接。当我们要访问远程位置的资源，而不是独立的页面时，XLink是变得更有趣。在上面的例子<description>元素集的XLINK属性显示的值为："new"。这意味着，应该在新窗口打开链接。我们可以设置XLINK中的值：显示属

性"embed"。这意味着资源应嵌入到页面处理。你认为这可能是另一个XML文档，而不是只是一个图像，你可以建立一个XML文档中层次结构的例子。

使用XLink，你还可以指定资源时才显示。这是由XLink的actuate属性处理。XLINK：actuate"="onLoad"指定的资源文件应加载和显示。XLINK：actuate="onRequest"意味着链接被点击之前无法读取或显示资源。这对低带宽设置非常方便。

XPointer 实例

让我们通过研究一个实例来学习一些基础的 XPointer 语法。

XPointer 实例

在本例中，我们会为您展示如何使用 XPointer 并结合 XLink 来指向另外一个文档的某个具体的部分。

我们将通过研究目标 XML 文档开始（即我们要链接的那个文档）。

目标XML文档

目标XML文档名为 "dogbreeds.xml"，它列出了一些不同的狗种类：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<dogbreeds>

<dog breed="Rottweiler" id="Rottweiler">
  <picture url="http://dog.com/rottweiler.gif" />
  <history>The Rottweiler's ancestors were probably Roman
  drover dogs.....</history>
  <temperament>Confident, bold, alert and imposing, the Rottweiler
  is a popular choice for its ability to protect....</temperament>
</dog>

<dog breed="FCRetriever" id="FCRetriever">
  <picture url="http://dog.com/fcretriever.gif" />
  <history>One of the earliest uses of retrieving dogs was to
  help fishermen retrieve fish from the water....</history>
  <temperament>The flat-coated retriever is a sweet, exuberant,
  lively dog that loves to play and retrieve....</temperament>
</dog>

</dogbreeds>
```

在您的浏览器查看 "dogbreeds.xml" 文件。

注意上面的 **XML** 文档在每个我们需要链接的元素上使用了 **id** 属性！

XML 链接文档

不止能够链接到整个文档（当使用 XLink 时），XPointer 允许您链接到文档的特定部分。如需链接到页面的某个具体的部分，请在 xlink:href 属性中的 URL 后添加一个井号 (#) 以及一个 XPointer 表达式。

表达式：*#xpointer(id("Rottweiler"))* 可引用目标文档中 id 值为 "Rottweiler" 的元素。

因此，xlink:href 属性会类似这样：*xlink:href="http://dog.com/dogbreeds.xml#xpointer(id('Rottweiler'))"*

不过，当使用 id 链接到某个元素时，XPointer 允许简写形式。您可以直接使用 id 的值，就像这样：*xlink:href="http://dog.com/dogbreeds.xml#Rottweiler"*。

下面的 XML 文档可引用每条狗的品种信息，均通过 XLink 和 XPointer 来引用：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<mydogs xmlns:xlink="http://www.w3.org/1999/xlink">

<mydog xlink:type="simple"
  xlink:href="http://dog.com/dogbreeds.xml#Rottweiler">
  <description xlink:type="simple"
  xlink:href="http://myweb.com/mydogs/anton.gif">
  Anton is my favorite dog. He has won a lot of.....
  </description>
</mydog>

<mydog xlink:type="simple"
  xlink:href="http://dog.com/dogbreeds.xml#FCRetriever">
  <description xlink:type="simple"
  xlink:href="http://myweb.com/mydogs/pluto.gif">
  Pluto is the sweetest dog on earth......
  </description>
</mydog>

</mydogs>
```

XLink 参考手册

XLink，即 XML 链接语言，是一种通过 W3C 推荐标准认证的 XML 标记语言，提供一些方法，用于在 XML 文件上创建内部和外部链接，以及与这些链接相关联的元数据。 XLink 提供两种可在 XML 文档中使用的超链接，简单链接和扩展链接。简单链接，只连接两种资源，类似于 HTML 链接和 IMG 链接。 扩展链接，可连接任意数量的资源。

XLink 属性参考手册

属性	值	描述
xlink:actuate	`onLoad` `onRequest` `other` `none`	定义何时读取和显示被链接的资源。
xlink:href	*URL*	要链接的 URL。
xlink:show	`embed` `new` `replace` `other` `none`	在何处打开链接。Replace 是默认值。
xlink:type	`simple` `extended` `locator` `arc` `resource` `title` `none`	链接的类型。

XML Schema 教程

XML Schema 简介

XML Schema 是基于 XML 的 DTD 替代者。

XML Schema 可描述 XML 文档的结构。

XML Schema 语言也可作为 XSD（XML Schema Definition）来引用。

您应当具备的基础知识

在继续学习之前，您需要对下面的知识有基本的了解：

- HTML / XHTML
- XML 以及 XML 命名空间
- 对 DTD 的基本了解

如果您希望首先学习这些项目，请在 首页 访问这些教程。

什么是 XML Schema？

XML Schema 的作用是定义 XML 文档的合法构建模块，类似 DTD。

XML Schema:

- 定义可出现在文档中的元素
- 定义可出现在文档中的属性
- 定义哪个元素是子元素
- 定义子元素的次序
- 定义子元素的数目
- 定义元素是否为空，或者是否可包含文本
- 定义元素和属性的数据类型
- 定义元素和属性的默认值以及固定值

XML Schema 是 DTD 的继任者

我们认为 XML Schema 很快会在大部分网络应用程序中取代 DTD。

理由如下：

- XML Schema 可针对未来的需求进行扩展
- XML Schema 更完善，功能更强大
- XML Schema 基于 XML 编写
- XML Schema 支持数据类型
- XML Schema 支持命名空间

XML Schema 是 W3C 标准

XML Schema 在 2001 年 5 月 2 日成为 W3C 标准。

您可以在我们的《W3C 教程》中获得更多有关 XML Schema 标准的信息。

为什么使用 XML Schemas?

XML Schema 比 DTD 更强大。

XML Schema 支持数据类型

XML Schema 最重要的能力之一就是对数据类型的支持。

通过对数据类型的支持：

- 可更容易地描述允许的文档内容
- 可更容易地验证数据的正确性
- 可更容易地与来自数据库的数据一并工作
- 可更容易地定义数据约束（data facets）
- 可更容易地定义数据模型（或称数据格式）
- 可更容易地在不同的数据类型间转换数据

编者注：数据约束，或称 facets，是 XML Schema 原型中的一个术语，中文可译为"面"，用来约束数据类型的容许值。

XML Schema 使用 XML 语法

另一个关于 XML Schema 的重要特性是，它们由 XML 编写。

由 XML 编写 XML Schema 有很多好处：

- 不必学习新的语言
- 可使用 XML 编辑器来编辑 Schema 文件
- 可使用 XML 解析器来解析 Schema 文件
- 可通过 XML DOM 来处理 Schema
- 可通过 XSLT 来转换 Schema

XML Schema 可保护数据通信

当数据从发送方被发送到接受方时，其要点是双方应有关于内容的相同的"期望值"。

通过 XML Schema，发送方可以用一种接受方能够明白的方式来描述数据。

一种数据，比如 "03-11-2004"，在某些国家被解释为11月3日，而在另一些国家为当作3月11日。

但是一个带有数据类型的 XML 元素，比如：<date type="date">2004-03-11</date>，可确保对内容一致的理解，这是因为 XML 的数据类型 "date" 要求的格式是 "YYYY-MM-DD"。

XML Schema 可扩展

XML Schema 是可扩展的，因为它们由 XML 编写。

通过可扩展的 Schema 定义，您可以：

- 在其他 Schema 中重复使用您的 Schema
- 创建由标准类型衍生而来的您自己的数据类型
- 在相同的文档中引用多重的 Schema

形式良好是不够的

我们把符合 XML 语法的文档称为形式良好的 XML 文档，比如：

- 它必须以 XML 声明开头
- 它必须拥有唯一的根元素
- 开始标签必须与结束标签相匹配
- 元素对大小写敏感
- 所有的元素都必须关闭
- 所有的元素都必须正确地嵌套
- 必须对特殊字符使用实体

即使文档的形式良好，仍然不能保证它们不会包含错误，并且这些错误可能会产生严重的后果。

请考虑下面的情况：您订购的了 5 打激光打印机，而不是 5 台。通过 XML Schema，大部分这样的错误会被您的验证软件捕获到。

XSD 如何使用?

XML 文档可对 DTD 或 XML Schema 进行引用。

一个简单的 XML 文档：

请看这个名为 "note.xml" 的 XML 文档：

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

DTD 文件

下面这个例子是名为 "note.dtd" 的 DTD 文件，它对上面那个 XML 文档（ "note.xml" ）的元素进行了定义：

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

第 1 行定义 note 元素有四个子元素："to, from, heading, body"。

第 2-5 行定义了 to, from, heading, body 元素的类型是 "#PCDATA"。

XML Schema

下面这个例子是一个名为 "note.xsd" 的 XML Schema 文件，它定义了上面那个 XML 文档（ "note.xml" ）的元素：

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

note 元素是一个复合类型，因为它包含其他的子元素。其他元素 (to, from, heading, body) 是简易类型，因为它们没有包含其他元素。您将在下面的章节学习更多有关复合类型和简易类型的知识。

对 DTD 的引用

此文件包含对 DTD 的引用：

```
<?xml version="1.0"?>

<!DOCTYPE note SYSTEM
"http://www.w3schools.com/dtd/note.dtd">

<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

对 XML Schema 的引用

此文件包含对 XML Schema 的引用：

```
<?xml version="1.0"?>

<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XSD - <schema> 元素

<schema> 元素是每一个 XML Schema 的根元素。

<schema> 元素

<schema> 元素是每一个 XML Schema 的根元素：

```
<?xml version="1.0"?>

<xs:schema>
...
...
</xs:schema>
```

<schema> 元素可包含属性。一个 schema 声明往往看上去类似这样：

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3school.cc"
xmlns="http://www.w3school.cc"
elementFormDefault="qualified">
...
...
</xs:schema>
```

以下代码片段:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

显示 schema 中用到的元素和数据类型来自命名空间 "http://www.w3.org/2001/XMLSchema"。同时它还规定了来自命名空间 "http://www.w3.org/2001/XMLSchema" 的元素和数据类型应该使用前缀 xs：

这个片断：

```
targetNamespace="http://www.w3school.cc"
```

显示被此 schema 定义的元素 (note, to, from, heading, body) 来自命名空间： "http://www.w3school.cc"。

这个片断：

```
xmlns="http://www.w3school.cc"
```

指出默认的命名空间是 "http://www.w3cschool.cc"。

这个片断：

```
elementFormDefault="qualified"
```

指出任何 XML 实例文档所使用的且在此 schema 中声明过的元素必须被命名空间限定。

在 XML 文档中引用 Schema

此 XML 文档含有对 XML Schema 的引用：

```
<?xml version="1.0"?>

<note xmlns="http://www.w3cschool.cc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3cschool.cc note.xsd">

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

下面的代码片断：

```
xmlns="http://www.w3cschool.cc"
```

规定了默认命名空间的声明。此声明会告知 schema 验证器，在此 XML 文档中使用的所有元素都被声明于 "http://www.w3cschool.cc" 这个命名空间。

一旦您拥有了可用的 XML Schema 实例命名空间：

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

您就可以使用 schemaLocation 属性了。此属性有两个值。第一个值是需要使用的命名空间。第二个值是供命名空间使用的 XML schema 的位置：

```
xsi:schemaLocation="http://www.w3cschool.cc note.xsd"
```

XSD 简易元素

XML Schema 可定义 XML 文件的元素。

简易元素指那些只包含文本的元素。它不会包含任何其他的元素或属性。

什么是简易元素？

简易元素指那些仅包含文本的元素。它不会包含任何其他的元素或属性。

不过，"仅包含文本"这个限定却很容易造成误解。文本有很多类型。它可以是 XML Schema 定义中包括的类型中的一种（布尔、字符串、数据等等），或者它也可以是您自行定义的定制类型。

您也可向数据类型添加限定（即 facets），以此来限制它的内容，或者您可以要求数据匹配某种特定的模式。

定义简易元素

定义简易元素的语法：

```
<xs:element name="xxx" type="yyy"/>
```

此处 xxx 指元素的名称，yyy 指元素的数据类型。XML Schema 拥有很多内建的数据类型。

最常用的类型是：

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

实例

这是一些 XML 元素：

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

这是相应的简易元素定义：

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

简易元素的默认值和固定值

简易元素可拥有指定的默认值或固定值。

当没有其他的值被规定时，默认值就会自动分配给元素。

在下面的例子中，缺省值是 "red"：

```
<xs:element name="color" type="xs:string" default="red"/>
```

固定值同样会自动分配给元素，并且您无法规定另外一个值。

在下面的例子中，固定值是 "red"：

```
<xs:element name="color" type="xs:string" fixed="red"/>
```

XSD 属性

所有的属性均作为简易类型来声明。

什么是属性？

简易元素无法拥有属性。假如某个元素拥有属性，它就会被当作某种复合类型。但是属性本身总是作为简易类型被声明的。

如何声明属性？

定义属性的语法是

```
<xs:attribute name="xxx" type="yyy"/>
```

在此处，xxx 指属性名称，yyy 则规定属性的数据类型。XML Schema 拥有很多内建的数据类型。

最常用的类型是：

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

实例

这是带有属性的 XML 元素：

```
<lastname lang="EN">Smith</lastname>
```

这是对应的属性定义：

```
<xs:attribute name="lang" type="xs:string"/>
```

属性的默认值和固定值

属性可拥有指定的默认值或固定值。

当没有其他的值被规定时，默认值就会自动分配给元素。

在下面的例子中，缺省值是 "EN"：

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
```

固定值同样会自动分配给元素，并且您无法规定另外的值。

在下面的例子中，固定值是 "EN"：

```
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

可选的和必需的属性

在缺省的情况下，属性是可选的。如需规定属性为必选，请使用 "use" 属性：

```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

对内容的限定

当 XML 元素或属性拥有被定义的数据类型时，就会向元素或属性的内容添加限定。

假如 XML 元素的类型是 "xs:date"，而其包含的内容是类似 "Hello World" 的字符串，元素将不会（通过）验证。

通过 XML schema，您也可向您的 XML 元素及属性添加自己的限定。这些限定被称为 facet（编者注：意为(多面体的)面，可译为限定面）。您会在下一节了解到更多有关 facet 的知识。

XSD 限定 / Facets

限定（restriction）用于为 XML 元素或者属性定义可接受的值。对 XML 元素的限定被称为 facet。

对值的限定

下面的例子定义了带有一个限定且名为 "age" 的元素。age 的值不能低于 0 或者高于 120：

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

对一组值的限定

如需把 XML 元素的内容限制为一组可接受的值，我们要使用枚举约束（enumeration constraint）。

下面的例子定义了带有一个限定的名为 "car" 的元素。可接受的值只有：Audi, Golf, BMW：

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

上面的例子也可以被写为：

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

注意： 在这种情况下，类型 "carType" 可被其他元素使用，因为它不是 "car" 元素的组成部分。

对一系列值的限定

如需把 XML 元素的内容限制定义为一系列可使用的数字或字母，我们要使用模式约束（pattern constraint）。

下面的例子定义了带有一个限定的名为 "letter" 的元素。可接受的值只有小写字母 a - z 其中的一个：

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下一个例子定义了带有一个限定的名为 "initials" 的元素。可接受的值是大写字母 A - Z 其中的三个：

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下一个例子也定义了带有一个限定的名为 "initials" 的元素。可接受的值是大写或小写字母 a - z 其中的三个：

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下一个例子定义了带有一个限定的名为 "choice 的元素。可接受的值是字母 x, y 或 z 中的一个：

```
<xs:element name="choice">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[xyz]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下一个例子定义了带有一个限定的名为 "prodid" 的元素。可接受的值是五个阿拉伯数字的一个序列，且每个数字的范围是 0-9：

```
<xs:element name="prodid">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

对一系列值的其他限定

下面的例子定义了带有一个限定的名为 "letter" 的元素。可接受的值是 a - z 中零个或多个字母：

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下面的例子定义了带有一个限定的名为 "letter" 的元素。可接受的值是一对或多对字母，每对字母由一个小写字母后跟一个大写字母组成。举个例子，"sToP"将会通过这种模式的验证，但是 "Stop"、"STOP" 或者 "stop" 无法通过验证：

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z][A-Z])+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下面的例子定义了带有一个限定的名为 "gender" 的元素。可接受的值是 male 或者 female：

```
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

下面的例子定义了带有一个限定的名为 "password" 的元素。可接受的值是由 8 个字符组成的一行字符，这些字符必须是大写或小写字母 a - z 亦或数字 0 - 9：

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

对空白字符的限定

如需规定对空白字符（whitespace characters）的处理方式，我们需要使用 whiteSpace 限定。

下面的例子定义了带有一个限定的名为 "address" 的元素。这个 whiteSpace 限定被设置为 "preserve"，这意味着 XML 处理器不会移除任何空白字符：

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

这个例子也定义了带有一个限定的名为 "address" 的元素。这个 whiteSpace 限定被设置为 "replace"，这意味着 XML 处理器将移除所有空白字符（换行、回车、空格以及制表符）：

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="replace"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

这个例子也定义了带有一个限定的名为 "address" 的元素。这个 whiteSpace 限定被设置为 "collapse"，这意味着 XML 处理器将移除所有空白字符（换行、回车、空格以及制表符会被替换为空格，开头和结尾的空格会被移除，而多个连续的空格会被缩减为一个单一的空格）：

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

对长度的限定

如需限制元素中值的长度，我们需要使用 length、maxLength 以及 minLength 限定。

本例定义了带有一个限定且名为 "password" 的元素。其值必须精确到 8 个字符：

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

这个例子也定义了带有一个限定的名为 "password" 的元素。其值最小为 5 个字符，最大为 8 个字符：

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

数据类型的限定

限定	描述
enumeration	定义可接受值的一个列表
fractionDigits	定义所允许的最大的小数位数。必须大于等于0。
length	定义所允许的字符或者列表项目的精确数目。必须大于或等于0。
maxExclusive	定义数值的上限。所允许的值必须小于此值。
maxInclusive	定义数值的上限。所允许的值必须小于或等于此值。
maxLength	定义所允许的字符或者列表项目的最大数目。必须大于或等于0。
minExclusive	定义数值的下限。所允许的值必需大于此值。
minInclusive	定义数值的下限。所允许的值必需大于或等于此值。
minLength	定义所允许的字符或者列表项目的最小数目。必须大于或等于0。
pattern	定义可接受的字符的精确序列。
totalDigits	定义所允许的阿拉伯数字的精确位数。必须大于0。
whiteSpace	定义空白字符（换行、回车、空格以及制表符）的处理方式。

XSD 复合元素

复合元素包含了其他的元素及/或属性。

h2>什么是复合元素？

复合元素指包含其他元素及/或属性的 XML 元素。

有四种类型的复合元素：

- 空元素
- 包含其他元素的元素
- 仅包含文本的元素
- 包含元素和文本的元素

注意： 上述元素均可包含属性！

复合元素的例子

复合元素，"product"，是空的：

```
<product pid="1345"/>
```

复合元素，"employee"，仅包含其他元素：

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

复合 XML 元素，"food"，仅包含文本：

```
<food type="dessert">Ice cream</food>
```

复合XML元素，"description"包含元素和文本：

```
<description>
It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

如何定义复合元素？

请看这个复合 XML 元素，"employee"，仅包含其他元素：

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

在 XML Schema 中，我们有两种方式来定义复合元素：

1. 通过命名此元素，可直接对"employee"元素进行声明，就像这样：

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

假如您使用上面所描述的方法，那么仅有 "employee" 可使用所规定的复合类型。请注意其子元素，"firstname" 以及 "lastname"，被包围在指示器 <sequence>中。这意味着子元素必须以它们被声明的次序出现。您会在 XSD 指示器 这一节学习更多有关指示器的知识。

2. "employee" 元素可以使用 type 属性，这个属性的作用是引用要使用的复合类型的名称：

```
<xs:element name="employee" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

如果您使用了上面所描述的方法，那么若干元素均可以使用相同的复合类型，比如这样：

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

您也可以在已有的复合元素之上以某个复合元素为基础，然后添加一些元素，就像这样：

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XSD 空元素

空的复合元素不能包含内容，只能含有属性。

复合空元素：

一个空的 XML 元素：

```
<product prodid="1345" />
```

上面的 "product" 元素根本没有内容。为了定义无内容的类型，我们就必须声明一个在其内容中只能包含元素的类型，但是实际上我们并不会声明任何元素，比如这样：

```
<xs:element name="product">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:integer">
        <xs:attribute name="prodid" type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

在上面的例子中，我们定义了一个带有复合内容的复合类型。complexContent 元素给出的信号是，我们打算限定或者拓展某个复合类型的内容模型，而 integer 限定则声明了一个属性但不会引入任何的元素内容。

但是，也可以更加紧凑地声明此 "product" 元素：

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

或者您可以为一个 complexType 元素起一个名字，然后为 "product" 元素设置一个 type 属性并引用这个 complexType 名称（通过使用此方法，若干个元素均可引用相同的复合类型）：

```
<xs:element name="product" type="prodtype"/>

<xs:complexType name="prodtype">
  <xs:attribute name="prodid" type="xs:positiveInteger"/>
</xs:complexType>
```

XSD 仅含元素

"仅含元素"的复合类型元素是只能包含其他元素的元素。

复合类型仅包含元素

XML 元素，"person"，仅包含其他的元素：

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

您可在 schema 中这样定义 "person" 元素：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

请留意这个 <xs:sequence>。它意味着被定义的元素必须按上面的次序出现在 "person" 元素中。</xs:sequence>

或者您可以为 complexType 元素设定一个名称，并让 "person" 元素的 type 属性来引用此名称（如使用此方法，若干元素均可引用相同的复合类型）：

```
<xs:element name="person" type="persontype"/>

<xs:complexType name="persontype">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

XSD 仅含文本

仅含文本的复合元素可包含文本和属性。

仅含文本的复合元素

此类型仅包含简易的内容（文本和属性），因此我们要向此内容添加 simpleContent 元素。当使用简易内容时，我们就必须在 simpleContent 元素内定义扩展或限定，就像这样：

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="basetype">
        ....
        ....
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

或者：

<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="basetype">
        ....
        ....
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

提示： 请使用 extension 或 restriction 元素来扩展或限制元素的基本简易类型。 这里有一个 XML 元素的例子，"shoesize"，其中仅包含文本：

```
<shoesize country="france">35</shoesize>
```

下面这个例子声明了一个复合类型，其内容被定义为整数值，并且 "shoesize" 元素含有名为 "country" 的属性：

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="country" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

我们也可为 complexType 元素设定一个名称，并让 "shoesize" 元素的 type 属性来引用此名称（通过使用此方法，若干元素均可引用相同的复合类型）：

```
<xs:element name="shoesize" type="shoetype"/>

<xs:complexType name="shoetype">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="country" type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

XSD 混合内容

混合的复合类型可包含属性、元素以及文本。

带有混合内容的复合类型

XML 元素，"letter"，含有文本以及其他元素：

```
<letter>
  Dear Mr.<name>John Smith</name>.
  Your order <orderid>1032</orderid>
  will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

下面这个 schema 声明了这个 "letter" 元素：

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

注意：为了使字符数据可以出现在 "letter" 的子元素之间，mixed 属性必须被设置为 "true"。<xs:sequence> 标签 (name、orderid 以及 shipdate) 意味着被定义的元素必须依次出现在 "letter" 元素内部。

我们也可以为 complexType 元素起一个名字，并让 "letter" 元素的 type 属性引用 complexType 的这个名称（通过这个方法，若干元素均可引用同一个复合类型）：

```
<xs:element name="letter" type="lettertype"/>

<xs:complexType name="lettertype" mixed="true">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="orderid" type="xs:positiveInteger"/>
    <xs:element name="shipdate" type="xs:date"/>
  </xs:sequence>
</xs:complexType>
```

XSD 指示器

通过指示器，我们可以控制在文档中使用元素的方式。

指示器

有七种指示器：

Order 指示器：

- All
- Choice
- Sequence

Occurrence 指示器：

- maxOccurs
- minOccurs

Group 指示器：

- Group name
- attributeGroup name

Order 指示器

Order 指示器用于定义元素的顺序。

All 指示器

<all> 指示器规定子元素可以按照任意顺序出现，且每个子元素必须只出现一次：

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

注意： 当使用 <all> 指示器时，你可以把 <minOccurs> 设置为 0 或者 1，而只能把 <maxOccurs> 指示器设置为 1（稍后将讲解 <minOccurs> 以及 <maxOccurs>）。

Choice 指示器

<choice> 指示器规定可出现某个子元素或者可出现另外一个子元素（非此即彼）：

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Sequence 指示器

<sequence> 规定子元素必须按照特定的顺序出现：

```
<xs:element name="person">
   <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Occurrence 指示器

Occurrence 指示器用于定义某个元素出现的频率。

注意： 对于所有的 "Order" 和 "Group" 指示器（any、all、choice、sequence、group name 以及 group reference），其中的 maxOccurs 以及 minOccurs 的默认值均为 1。

maxOccurs 指示器

<maxOccurs> 指示器可规定某个元素可出现的最大次数：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

上面的例子表明，子元素 "child_name" 可在 "person" 元素中最少出现一次（其中 minOccurs 的默认值是 1），最多出现 10 次。

minOccurs 指示器

<minOccurs> 指示器可规定某个元素能够出现的最小次数：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

上面的例子表明，子元素 "child_name" 可在 "person" 元素中出现最少 0 次，最多出现 10 次。

提示：如需使某个元素的出现次数不受限制，请使用 maxOccurs="unbounded" 这个声明：

一个实际的例子：

名为 "Myfamily.xml" 的 XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="family.xsd">

<person>
  <full_name>Hege Refsnes</full_name>
  <child_name>Cecilie</child_name>
</person>

<person>
  <full_name>Tove Refsnes</full_name>
  <child_name>Hege</child_name>
  <child_name>Stale</child_name>
  <child_name>Jim</child_name>
  <child_name>Borge</child_name>
</person>

<person>
  <full_name>Stale Refsnes</full_name>
</person>

</persons>
```

上面这个 XML 文件含有一个名为 "persons" 的根元素。在这个根元素内部，我们定义了三个 "person" 元素。每个 "person" 元素必须含有一个 "full_name" 元素，同时它可以包含多至 5 个 "child_name" 元素。

这是schema文件"family.xsd"：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

<xs:element name="persons">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="person" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="full_name" type="xs:string"/>
            <xs:element name="child_name" type="xs:string"
            minOccurs="0" maxOccurs="5"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Group 指示器

Group 指示器用于定义相关的数批元素。

元素组

元素组通过 group 声明进行定义：

```
<xs:group name="groupname">
...
</xs:group>
```

您必须在 group 声明内部定义一个 all、choice 或者 sequence 元素。下面这个例子定义了名为 "persongroup" 的 group，它定义了必须按照精确的顺序出现的一组元素：

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>
```

在您把 group 定义完毕以后，就可以在另一个定义中引用它了：

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>

<xs:element name="person" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

属性组

属性组通过 attributeGroup 声明来进行定义：

```
<xs:attributeGroup name="groupname">
...
</xs:attributeGroup>
```

下面这个例子定义了名为 "personattrgroup" 的一个属性组：

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
```

在您已定义完毕属性组之后，就可以在另一个定义中引用它了，就像这样：

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>

<xs:element name="person">
  <xs:complexType>
    <xs:attributeGroup ref="personattrgroup"/>
  </xs:complexType>
</xs:element>
```

XSD <any> 元素

><any> 元素使我们有能力通过未被 schema 规定的元素来拓展 XML 文档！

h2> 元素

<any> 元素使我们有能力通过未被 schema 规定的元素来拓展 XML 文档！

下面这个例子是从名为 "family.xsd" 的 XML schema 中引用的片段。它展示了一个针对 "person" 元素的声明。通过使用 <any> 元素，我们可以通过任何元素（在 <lastname> 之后）扩展 "person" 的内容：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

现在，我们希望使用 "children" 元素来扩展 "person" 元素。这此种情况下我们就可以这么做，即使以上这个 schema 的作者没有声明任何 "children" 元素。

请看这个 schema 文件，名为 "children.xsd"：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="children">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="childname" type="xs:string"
      maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

下面这个 XML 文件（名为 "Myfamily.xml"），使用了来自两个不同的 schema 中的成分，"family.xsd" 和 "children.xsd"：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<persons xmlns="http://www.microsoft.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.microsoft.com family.xsd
http://www.w3schools.com children.xsd">

<person>
  <firstname>Hege</firstname>
  <lastname>Refsnes</lastname>
  <children>
    <childname>Cecilie</childname>
  </children>
</person>

<person>
  <firstname>Stale</firstname>
  <lastname>Refsnes</lastname>
</person>

</persons>
```

上面这个 XML 文件是有效的，这是由于 schema "family.xsd" 允许我们通过在 "lastname" 元素后的可选元素来扩展 "person" 元素。

<any> 和 <anyAttribute> 均可用于制作可扩展的文档！它们使文档有能力包含未在主 XML schema 中声明过的附加元素。

XSD <anyAttribute> 元素

<anyAttribute> 元素使我们有能力通过未被 schema 规定的属性来扩展 XML 文档！

<anyAttribute> 元素使我们有能力通过未被 schema 规定的属性来扩展 XML 文档！

下面的例子是来自名为 "family.xsd" 的 XML schema 的一个片段。它为我们展示了针对 "person" 元素的一个声明。通过使用 <anyAttribute> 元素，我们就可以向 "person" 元素添加任意数量的属性：

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

现在，我们希望通过 "gender" 属性来扩展 "person" 元素。在这种情况下我们就可以这样做，即使这个 schema 的作者从未声明过任何 "gender" 属性。

请看这个 schema 文件，名为 "attribute.xsd"：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:attribute name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

</xs:schema>
```

下面这个 XML（名为 "Myfamily.xml"），使用了来自不同 schema 的成分，"family.xsd" 和 "attribute.xsd"：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<persons xmlns="http://www.microsoft.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:SchemaLocation="http://www.microsoft.com family.xsd
http://www.w3schools.com attribute.xsd">

<person gender="female">
  <firstname>Hege</firstname>
  <lastname>Refsnes</lastname>
</person>

<person gender="male">
  <firstname>Stale</firstname>
  <lastname>Refsnes</lastname>
</person>

</persons>
```

上面这个 XML 文件是有效的，这是因为 schema "family.xsd" 允许我们向 "person" 元素添加属性。

<any> 和 <anyAttribute> 均可用于制作可扩展的文档！它们使文档有能力包含未在主 XML schema 中声明过的附加元素。

XSD 元素替换(Element Substitution)

通过 XML Schema，一个元素可对另一个元素进行替换。

元素替换

让我们举例说明：我们的用户来自英国和挪威。我们希望有能力让用户选择在 XML 文档中使用挪威语的元素名称还是英语的元素名称。

为了解决这个问题，我们可以在 XML schema 中定义一个 *substitutionGroup*。首先，我们声明主元素，然后我们会声明次元素，这些次元素可声明它们能够替换主元素。

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>
```

在上面的例子中，"name" 元素是主元素，而 "navn" 元素可替代 "name" 元素。

请看一个 XML schema 的片段：

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>

<xs:complexType name="custinfo">
  <xs:sequence>
    <xs:element ref="name"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="customer" type="custinfo"/>
<xs:element name="kunde" substitutionGroup="customer"/>
```

有效的 XML 文档类似这样（根据上面的 schema）：

```
<customer>
  <name>John Smith</name>
</customer>
```

或类似这样：

```
<kunde>
  <navn>John Smith</navn>
</kunde>
```

阻止元素替换

为防止其他的元素替换某个指定的元素，请使用 block 属性：

```
<xs:element name="name" type="xs:string" block="substitution"/>
```

请看某个 XML schema 的片段：

```
<xs:element name="name" type="xs:string" block="substitution"/>
<xs:element name="navn" substitutionGroup="name"/>

<xs:complexType name="custinfo">
  <xs:sequence>
    <xs:element ref="name"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="customer" type="custinfo" block="substitution"/>
<xs:element name="kunde" substitutionGroup="customer"/>
```

合法的 XML 文档应该类似这样（根据上面的 schema）：

```
<customer>
  <name>John Smith</name>
</customer>
```

但是下面的文档不再合法：

```
<kunde>
  <navn>John Smith</navn>
</kunde>
```

使用 substitutionGroup

可替换元素的类型必须和主元素相同，或者从主元素衍生而来。假如可替换元素的类型与主元素的类型相同，那么您就不必规定可替换元素的类型了。

请注意，substitutionGroup 中的所有元素（主元素和可替换元素）必须被声明为全局元素，否则就无法工作！

什么是全局元素（Global Elements）？

全局元素指 "schema" 元素的直接子元素！本地元素（Local elements）指嵌套在其他元素中的元素。

XSD 实例

本节会为您演示如何编写一个 XML Schema。您还将学习到编写 schema 的不同方法。

XML 文档

让我们看看这个名为 "shiporder.xml" 的 XML 文档：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

上面的XML文档包括根元素 "shiporder"，其中包含必须名为 "orderid" 的属性。"shiporder" 元素包含三个不同的子元素："orderperson"、"shipto" 以及 "item"。"item" 元素出现了两次，它含有一个 "title"、一个可选 "note" 元素、一个 "quantity" 以及一个 "price" 元素。

上面这一行 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"，告知XML解析器根据某个 schema 来验证此文档。这一行：xsi:noNamespaceSchemaLocation="shiporder.xsd" 规定了 schema 的位置（在这里，它与 "shiporder.xml" 处于相同的文件夹）。

创建一个 XML Schema

现在，我们需要为上面这个 XML 文档创建一个 schema。

我们可以通过打开一个新的文件来开始，并把这个文件命名为 "shiporder.xsd"。要创建 schema，我们仅仅需要简单地遵循 XML 文档中的结构，定义我们所发现的每个元素。首先我们开始定义一个标准的 XML 声明：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

在上面的 schema 中，我们使用了标准的命名空间 (xs)，与此命名空间相关联的 URI 是 Schema 的语言定义（Schema language definition），其标准值是 http://www.w3.org/2001/XMLSchema。

接下来，我们需要定义 "shiporder" 元素。此元素拥有一个属性，其中包含其他的元素，因此我们将它认定为复合类型。"shiporder" 元素的子元素被 xs:sequence 元素包围，定义了子元素的次序：

```
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

然后我们需要把 "orderperson" 元素定义为简易类型（这是因为它不包含任何属性或者其他的元素）。类型 (xs:string) 的前缀是由命名空间的前缀规定的，此命名空间与指示预定义的 schema 数据类型的 XML schema 相关联：

```
<xs:element name="orderperson" type="xs:string"/>
```

接下来，我需要把两个元素定义为复合类型："shipto" 和 "item"。我们从定义 "shipto" 元素开始：

```
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

通过 schema，我们可使用 maxOccurs 和 minOccurs 属性来定义某个元素可能出现的次数。maxOccurs 定义某元素出现次数的最大值，而 minOccurs 则定义某元素出现次数的最小值。maxOccurs 和 minOccurs 的默认值都是 1！

现在，我们可以定义 "item" 元素了。这个元素可在 "shiporder" 元素内部出现多次。这是通过把 "item" 元素的 maxOccurs 属性的值设定为 "unbounded" 来实现的，这样 "item" 元素就可出现创作者所希望的任意多次。请注意，"note" 元素是可选元素。我们已经把此元素的 minOccurs 属性设定为 0 了：

```
<xs:element name="item" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="note" type="xs:string" minOccurs="0"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="price" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

现在，我们可以声明 "shiporder" 元素的属性了。由于这是一个必选属性，我们规定 use="required"。

注意：此属性的声明必须被置于最后：

```
<xs:attribute name="orderid" type="xs:string" use="required"/>
```

这是这个名为 "shiporder.xsd" 的 schema 文件的文档清单：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

分割 Schema

前面的设计方法非常容易，但当文档很复杂时却难以阅读和维护。

接下来介绍的设计方法基于首先对所有元素和属性的定义，然后再使用 ref 属性来引用它们。

这是用新方法设计的 schema 文件("shiporder.xsd")：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- definition of simple elements -->
<xs:element name="orderperson" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="country" type="xs:string"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="note" type="xs:string"/>
<xs:element name="quantity" type="xs:positiveInteger"/>
<xs:element name="price" type="xs:decimal"/>

<!-- definition of attributes -->
<xs:attribute name="orderid" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="address"/>
      <xs:element ref="city"/>
      <xs:element ref="country"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="note" minOccurs="0"/>
      <xs:element ref="quantity"/>
      <xs:element ref="price"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="orderperson"/>
      <xs:element ref="shipto"/>
      <xs:element ref="item" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="orderid" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

使用指定的类型（Named Types）

第三种设计方法定义了类或者类型，这样使我们有能力重复使用元素的定义。具体的方式是：首先对简易元素和复合元素进行命名，然后通过元素的 type 属性来指向它们

这是利用第三种方法设计的 schema 文件 ("shiporder.xsd")：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:simpleType name="stringtype">
  <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:simpleType name="inttype">
  <xs:restriction base="xs:positiveInteger"/>
</xs:simpleType>

<xs:simpleType name="dectype">
  <xs:restriction base="xs:decimal"/>
</xs:simpleType>

<xs:simpleType name="orderidtype">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]{6}"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="shiptotype">
  <xs:sequence>
    <xs:element name="name" type="stringtype"/>
    <xs:element name="address" type="stringtype"/>
    <xs:element name="city" type="stringtype"/>
    <xs:element name="country" type="stringtype"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="itemtype">
  <xs:sequence>
    <xs:element name="title" type="stringtype"/>
    <xs:element name="note" type="stringtype" minOccurs="0"/>
    <xs:element name="quantity" type="inttype"/>
    <xs:element name="price" type="dectype"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="shipordertype">
  <xs:sequence>
    <xs:element name="orderperson" type="stringtype"/>
    <xs:element name="shipto" type="shiptotype"/>
    <xs:element name="item" maxOccurs="unbounded" type="itemtype"/>
  </xs:sequence>
  <xs:attribute name="orderid" type="orderidtype" use="required"/>
</xs:complexType>

<xs:element name="shiporder" type="shipordertype"/>

</xs:schema>
```

restriction 元素显示出数据类型源自于 W3C XML Schema 命名空间的数据类型。因此，下面的片段也就意味着元素或属性的值必须是字符串类型的值：

```
<xs:restriction base="xs:string">
```

restriction 元素常被用于向元素施加限制。请看下面这些来自以上 schema 的片段：

```
<xs:simpleType name="orderidtype">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]{6}"/>
  </xs:restriction>
</xs:simpleType>
```

这段代码指示出，元素或属性的值必须为字符串，并且必须是连续的六个字符，同时这些字符必须是 0-9 的数字。

XSD 字符串 数据类型

字符串数据类型用于可包含字符串的值。

字符串数据类型（String Data Type）

字符串数据类型可包含字符、换行、回车以及制表符。

下面是一个关于某个 scheme 中字符串声明的例子：

```
<xs:element name="customer" type="xs:string"/>
```

文档中的元素看上去应该类似这样：

```
<customer>John Smith</customer>
```

或者类似这样：

```
<customer>       John Smith     </customer>
```

注意：如果您使用字符串数据类型，XML 处理器就不会更改其中的值。

规格化字符串数据类型（NormalizedString Data Type）

规格化字符串数据类型源自于字符串数据类型。

规格化字符串数据类型同样可包含字符，但是 XML 处理器会移除折行，回车以及制表符。

下面是一个关于在某个 schema 中规格化字符串数据类型的例子：

```
<xs:element name="customer" type="xs:normalizedString"/>
```

文档中的元素看上去应该类似这样：

```
<customer>John Smith</customer>
```

或者类似这样：

```
<customer>     John Smith     </customer>
```

注意：在上面的例子中，XML 处理器会使用空格替换所有的制表符。

Token 数据类型（Token Data Type）

Token 数据类型同样源自于字符串数据类型。

Token 数据类型同样可包含字符，但是 XML 处理器会移除换行符、回车、制表符、开头和结尾的空格以及（连续的）空格。

下面是在 schema 中一个有关 token 声明的例子：

```
<xs:element name="customer" type="xs:token"/>
```

文档中的元素看上去应该类似这样：

```
<customer>John Smith</customer>
```

或者类似这样：

```
<customer>     John Smith     </customer>
```

注意：>在上面这个例子中，XML 解析器会移除制表符。

字符串数据类型

请注意，所有以下的数据类型均衍生于字符串数据类型（除了字符串数据类型本身）！

名称	描述
ENTITIES	
ENTITY	
ID	在 XML 中提交 ID 属性的字符串 (仅与 schema 属性一同使用)
IDREF	在 XML 中提交 IDREF 属性的字符串(仅与 schema 属性一同使用)
IDREFS language	包含合法的语言 id 的字符串
Name	包含合法 XML 名称的字符串
NCName	
NMTOKEN	在 XML 中提交 NMTOKEN 属性的字符串 (仅与 schema 属性一同使用)
NMTOKENS	
normalizedString	不包含换行符、回车或制表符的字符串
QName	
string	字符串
token	不包含换行符、回车或制表符、开头或结尾空格或者多个连续空格的字符串

对字符串数据类型的限定（Restriction）

可与字符串数据类型一同使用的限定：

- enumeration
- length
- maxLength
- minLength
- pattern (NMTOKENS、IDREFS 以及 ENTITIES 无法使用此约束)
- whiteSpace

XSD 日期和时间数据类型

日期及时间数据类型用于包含日期和时间的值。

日期数据类型（Date Data Type）

日期数据类型用于定义日期。

日期使用此格式进行定义："YYYY-MM-DD"，其中：

- YYYY 表示年份
- MM 表示月份
- DD 表示天数

注意：所有的成分都是必需的

下面是一个有关 schema 中日期声明的例子：

```
<xs:element name="start" type="xs:date"/>
```

文档中的元素看上去应该类似这样：

```
<start>2002-09-24</start>
```

时区

如需规定一个时区，您也可以通过在日期后加一个 "Z" 的方式，使用世界调整时间（UTC time）来输入一个日期 - 比如这样：

```
<start>2002-09-24Z</start>
```

或者也可以通过在日期后添加一个正的或负时间的方法，来规定以世界调整时间为准的偏移量 - 比如这样：

```
<start>2002-09-24-06:00</start>

或者

<start>2002-09-24+06:00</start>
```

h2>时间数据类型（Time Data Type）

时间数据类型用于定义时间。

时间使用下面的格式来定义："hh:mm:ss"，其中

- hh 表示小时
- mm 表示分钟
- ss 表示秒

注意：所有的成分都是必需的！

下面是一个有关 schema 中时间声明的例子：

```
<xs:element name="start" type="xs:time"/>
```

文档中的元素看上去应该类似这样：

```
<start>09:00:00</start>
```

或者类似这样：

```
<start>09:30:10.5</start>
```

时区

如需规定一个时区，您也可以通过在时间后加一个 "Z" 的方式，使用世界调整时间（UTC time）来输入一个时间 - 比如这样：

```
<start>09:30:10Z</start>
```

或者也可以通过在时间后添加一个正的或负时间的方法，来规定以世界调整时间为准的偏移量 - 比如这样：

```
<start>09:30:10-06:00</start>

or

<start>09:30:10+06:00</start>
```

日期时间数据类型（DateTime Data Type）

日期时间数据类型用于定义日期和时间。

日期时间使用下面的格式进行定义："YYYY-MM-DDThh:mm:ss"，其中：

- YYYY 表示年份
- MM 表示月份
- DD 表示日
- T 表示必需的时间部分的起始
- hh 表示小时
- mm 表示分钟
- ss 表示秒

注意： 所有的成分都是必需的！

下面是一个有关 schema 中日期时间声明的例子：

```
<xs:element name="startdate" type="xs:dateTime"/>
```

文档中的元素看上去应该类似这样：

```
<startdate>2002-05-30T09:00:00</startdate>
```

或者类似这样：

```
<startdate>2002-05-30T09:30:10.5</startdate>
```

时区

如需规定一个时区，您也可以通过在日期时间后加一个 "Z" 的方式，使用世界调整时间（UTC time）来输入一个日期时间 - 比如这样：

```
<startdate>2002-05-30T09:30:10Z</startdate>
```

或者也可以通过在时间后添加一个正的或负时间的方法，来规定以世界调整时间为准的偏移量 - 比如这样：

```
<startdate>2002-05-30T09:30:10-06:00</startdate>

或者

<startdate>2002-05-30T09:30:10+06:00</startdate>
```

h2>持续时间数据类型（Duration Data Type）

持续时间数据类型用于规定时间间隔。

时间间隔使用下面的格式来规定："PnYnMnDTnHnMnS"，其中：

- P 表示周期(必需)

- nY 表示年数
- nM 表示月数
- nD 表示天数
- T 表示时间部分的起始 （如果您打算规定小时、分钟和秒，则此选项为必需）
- nH 表示小时数
- nM 表示分钟数
- nS 表示秒数

下面是一个有关 schema 中持续时间声明的例子：

```
<xs:element name="period" type="xs:duration"/>
```

文档中的元素看上去应该类似这样：

```
<period>P5Y</period>
```

上面的例子表示一个 5 年的周期。

或者类似这样：

```
<period>P5Y2M10D</period>
```

上面的例子表示一个 5 年、2 个月及 10 天的周期。

或者类似这样：

```
<period>P5Y2M10DT15H</period>
```

上面的例子表示一个 5 年、2 个月、10 天及 15 小时的周期。

或者类似这样：

```
<period>PT15H</period>
```

上面的例子表示一个 15 小时的周期。

负的持续时间

如需规定一个负的持续时间，请在 P 之前输入减号：

```
<period>-P10D</period>
```

上面的例子表示一个负 10 天的周期。

日期和时间数据类型

名称	描述
date	定义一个日期值
dateTime	定义一个日期和时间值
duration	定义一个时间间隔
gDay	定义日期的一个部分 - 天 (DD)
gMonth	定义日期的一个部分 - 月 (MM)
gMonthDay	定义日期的一个部分 - 月和天 (MM-DD)
gYear	定义日期的一个部分 - 年 (YYYY)
gYearMonth	定义日期的一个部分 - 年和月 (YYYY-MM)
time	定义一个时间值

对日期数据类型的限定（Restriction）

可与日期数据类型一同使用的限定：

- enumeration
- maxExclusive
- maxInclusive
- minExclusive
- minInclusive
- pattern
- whiteSpace

XSD 数值数据类型

十进制数据类型

十进制数据类型用于规定一个数值。

下面是一个关于某个 scheme 中十进制数声明的例子。

```
<xs:element name="prize" type="xs:decimal"/>
```

文档中的元素看上去应该类似这样：

```
<prize>999.50</prize>
```

或者类似这样：

```
<prize>+999.5450</prize>
```

或者类似这样：

```
<prize>-999.5230</prize>
```

或者类似这样：

```
<prize>0</prize>
```

或者类似这样：

```
<prize>14</prize>
```

注意： 您可规定的十进制数字的最大位数是 18 位。

整数数据类型

整数数据类型用于规定无小数成分的数值。

下面是一个关于某个 scheme 中整数声明的例子。

```
<xs:element name="prize" type="xs:integer"/>
```

文档中的元素看上去应该类似这样：

```
<prize>999</prize>
```

或者类似这样：

```
<prize>+999</prize>
```

或者类似这样：

```
<prize>-999</prize>
```

或者类似这样：

```
<prize>0</prize>
```

数值数据类型

请注意，下面所有的数据类型均源自于十进制数据类型（除 decimal 本身以外）！

名字	秒数
byte	有正负的 8 位整数
decimal	十进制数
int	有正负的 32 位整数
integer	整数值
long	有正负的 64 位整数
negativeInteger	仅包含负值的整数 (.., -2, -1.)
nonNegativeInteger	仅包含非负值的整数 (0, 1, 2, ..)
nonPositiveInteger	仅包含非正值的整数 (.., -2, -1, 0)
positiveInteger	仅包含正值的整数 (1, 2, ..)
short	有正负的 16 位整数
unsignedLong	无正负的 64 位整数
unsignedInt	无正负的 32 位整数
unsignedShort	无正负的 16 位整数
unsignedByte	无正负的 8 位整数

对数值数据类型的限定（Restriction）

可与数值数据类型一同使用的限定：

- enumeration
- fractionDigits
- maxExclusive
- maxInclusive
- minExclusive
- minInclusive
- pattern
- totalDigits
- whiteSpace

XSD 杂项 数据类型

其他杂项数据类型包括布尔、base64Binary、十六进制、浮点、双精度、anyURI、anyURI 以及 NOTATION。

布尔数据类型（Boolean Data Type）

布尔数据性用于规定 true 或 false 值。

下面是一个关于某个 scheme 中逻辑声明的例子：

```
<xs:attribute name="disabled" type="xs:boolean"/>
```

文档中的元素看上去应该类似这样：

```
<prize disabled="true">999</prize>
```

注意： 合法的布尔值是 true、false、1（表示 true） 以及 0（表示 false）。

二进制数据类型（Binary Data Types）

二进制数据类型用于表达二进制形式的数据。

我们可使用两种二进制数据类型：

- base64Binary (Base64 编码的二进制数据)
- hexBinary (十六进制编码的二进制数据)

下面是一个关于某个 scheme 中 hexBinary 声明的例子：

```
<xs:element name="blobsrc" type="xs:hexBinary"/>
```

AnyURI 数据类型（AnyURI Data Type）

anyURI 数据类型用于规定 URI。

下面是一个关于某个 scheme 中 anyURI 声明的例子：

```
<xs:attribute name="src" type="xs:anyURI"/>
```

文档中的元素看上去应该类似这样：

```
<pic src="http://www.w3schools.com/images/smiley.gif" />
```

注意： 如果某个 URI 含有空格，请用 %20 替换它们。

杂项数据类型

名称	描述
anyURI	
base64Binary	
boolean	
double	
float	
hexBinary	
NOTATION	
QName	

对杂项数据类型的限定（Restriction）

可与杂项数据类型一同使用的限定：

- enumeration (布尔数据类型无法使用此约束*)
- length (布尔数据类型无法使用此约束)
- maxLength (布尔数据类型无法使用此约束)
- minLength (布尔数据类型无法使用此约束)
- pattern
- whiteSpace

*译者注：约束指 constraint。

XML 编辑器

使用专业的XML编辑器你可以更好的编写XML文档。

XML 是基于文本的

XML 指可扩展标记语言

你可以使用简单的XML编辑器如记事本来创建或者编辑XML文件。

然而，当你开始使用XML，你很快就会发现它使用专业的XML编辑器可以更好的 编辑XML文件。

为什么不使用记事本?

许多开发人员使用记事本来编辑 XML 和 HTML 文档，这是因为最常用的操作系统都带有记事本，而且它很容易使用。从个人来讲，我经常使用记事本来快速地编辑某些简单的 HTML、CSS 以及 XML 文件。

但是，如果您使用记事本对 XML 进行编辑，可能很快会发现不少问题。

不能确定您编辑的文档类型，所以也就无法辅助您的工作。

为什么使用 XML 编辑器?

当今，XML 是非常重要的技术，并且开发项目正在使用这些基于 XML 的技术：

- 用 XML Schema 定义 XML 的结构和数据类型
- 用 XSLT 来转换 XML 数据
- 用 SOAP 来交换应用程序之间的 XML 数据
- 用 WSDL 来描述网络服务
- 用 RDF 来描述网络资源
- 用 XPath 和 XQuery 来访问 XML 数据
- 用 SMIL 来定义图形

为了能够编写出无错的 XML 文档，您需要一款智能的 XML 编辑器！

XML 编辑器

专业的 XML 编辑器会帮助您编写无错的 XML 文档，根据某种 DTD 或者 schema 来验证 XML，以及强制您创建合法的 XML 结构。

XML 编辑器应该具有如下能力：

- 为开始标签自动添加结束标签
- 强制您编写合法的 XML
- 根据 DTD 来验证 XML
- 根据 Schema 来验证 XML
- 对您的 XML 语法进行代码的颜色化显示

XML Schema 参考手册

XSD 元素

元素	解释
all	规定子元素能够以任意顺序出现，每个子元素可出现零次或一次。
annotation	annotation 元素是一个顶层元素，规定 schema 的注释。
any	使创作者可以通过未被 schema 规定的元素来扩展 XML 文档。
anyAttribute	使创作者可以通过未被 schema 规定的属性来扩展 XML 文档。
appInfo	规定 annotation 元素中应用程序要使用的信息。
attribute	定义一个属性。
attributeGroup	定义在复杂类型定义中使用的属性组。
choice	仅允许在 <choice> 声明中包含一个元素出现在包含元素中。
complexContent	定义对复杂类型（包含混合内容或仅包含元素）的扩展或限制。
complexType	定义复杂类型。
documentation	定义 schema 中的文本注释。
element	定义元素。
extension	扩展已有的 simpleType 或 complexType 元素。
field	规定 XPath 表达式，该表达式规定用于定义标识约束的值。
group	定义在复杂类型定义中使用的元素组。
import	向一个文档添加带有不同目标命名空间的多个 schema。
include	向一个文档添加带有相同目标命名空间的多个 schema。
key	指定属性或元素值（或一组值）必须是指定范围内的键。
keyref	规定属性或元素值（或一组值）对应指定的 key 或 unique 元素的值。
list	把简单类型定义为指定数据类型的值的一个列表。
notation	描述 XML 文档中非 XML 数据的格式。
redefine	重新定义从外部架构文件中获取的简单和复杂类型、组和属性组。
restriction	定义对 simpleType、simpleContent 或 complexContent 的约束。
schema	定义 schema 的根元素。
selector	指定 XPath 表达式，该表达式为标识约束选择一组元素。
sequence	要求子元素必须按顺序出现。每个子元素可出现 0 到任意次数。

simpleType	定义一个简单类型，规定约束以及关于属性或仅含文本的元素的值的信息。
union	定义多个 simpleType 定义的集合。
unique	指定属性或元素值（或者属性或元素值的组合）在指定范围内必须是唯一的。

XSD 限定/Facets

参阅 XSD 限定 / Facets

限定	描述
enumeration	定义可接受值的一个列表
fractionDigits	定义所允许的最大的小数位数。必须大于等于0。
length	定义所允许的字符或者列表项目的精确数目。必须大于或等于0。
maxExclusive	定义数值的上限。所允许的值必须小于此值。
maxInclusive	定义数值的上限。所允许的值必须小于或等于此值。
maxLength	定义所允许的字符或者列表项目的最大数目。必须大于或等于0。
minExclusive	定义数值的下限。所允许的值必需大于此值。
minInclusive	定义数值的下限。所允许的值必需大于或等于此值。
minLength	定义所允许的字符或者列表项目的最小数目。必须大于或等于0。
pattern	定义可接受的字符的精确序列。
totalDigits	定义所允许的阿拉伯数字的精确位数。必须大于0。
whiteSpace	定义空白字符（换行、回车、空格以及制表符）的处理方式。

SOAP 教程

SOAP 简介

SOAP 是基于 XML 的简易协议，可使应用程序在 HTTP 之上进行信息交换。

或者更简单地说：SOAP 是用于访问网络服务的协议。

您应当具备的基础知识

在继续学习之前，您需要对下面的知识有基本的了解：

- XML
- XML 命名空间

如果您希望首先学习这些项目，请访问我们的《XML 教程》。

什么是 SOAP？

- SOAP 指简易对象访问协议
- SOAP 是一种通信协议
- SOAP 用于应用程序之间的通信
- SOAP 是一种用于发送消息的格式
- SOAP 被设计用来通过因特网进行通信
- SOAP 独立于平台
- SOAP 独立于语言
- SOAP 基于 *XML*
- SOAP 很简单并可扩展
- SOAP 允许您绕过防火墙
- SOAP 将被作为 *W3C* 标准来发展

为什么使用 SOAP？

对于应用程序开发来说，使程序之间进行因特网通信是很重要的。

目前的应用程序通过使用远程过程调用（RPC）在诸如 DCOM 与 CORBA 等对象之间进行通信，但是 HTTP 不是为此设计的。RPC 会产生兼容性以及安全问题；防火墙和代理服务器通常会阻止此类流量。

通过 HTTP 在应用程序间通信是更好的方法，因为 HTTP 得到了所有的因特网浏览器及服务器的支持。SOAP 就是被创造出来完成这个任务的。

SOAP 提供了一种标准的方法，使得运行在不同的操作系统并使用不同的技术和编程语言的应用程序可以互相进行通信。

Microsoft 和 SOAP

SOAP 是微软 .net 架构的关键元素，用于未来的因特网应用程序开发。

SOAP 1.1 被提交到 W3C

在 2000 年 5 月，UserLand、Ariba、Commerce One、Compaq、Developmentor、HP、IBM、IONA、Lotus、Microsoft 以及 SAP 向 W3C 提交了 SOAP 因特网协议，这些公司期望此协议能够通过使用因特网标准（HTTP 以及 XML）把图形用户界面桌面应用程序连接到强大的因特网服务器，以此来彻底变革应用程序的开发。

W3C 正在发展 SOAP 1.2

首个关于 SOAP 的公共工作草案由 W3C 在 2001 年 12 月发布。如需阅读更多有关在 W3C 的 SOAP 活动的内容，请访问我们的《W3C 教程》。

SOAP 语法

SOAP 构建模块

一条 SOAP 消息就是一个普通的 XML 文档，包含下列元素：

- 必需的 Envelope 元素，可把此 XML 文档标识为一条 SOAP 消息
- 可选的 Header 元素，包含头部信息
- 必需的 Body 元素，包含所有的调用和响应信息
- 可选的 Fault 元素，提供有关在处理此消息所发生错误的信息

所有以上的元素均被声明于针对 SOAP 封装的默认命名空间中：

http://www.w3.org/2001/12/soap-envelope

以及针对 SOAP 编码和数据类型的默认命名空间：

http://www.w3.org/2001/12/soap-encoding

语法规则

这里是一些重要的语法规则：

- SOAP 消息必须用 XML 来编码
- SOAP 消息必须使用 SOAP Envelope 命名空间
- SOAP 消息必须使用 SOAP Encoding 命名空间
- SOAP 消息不能包含 DTD 引用
- SOAP 消息不能包含 XML 处理指令

SOAP 消息的基本结构

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

SOAP Envelope 元素

强制使用的 SOAP 的 Envelope 元素是 SOAP 消息的根元素。

SOAP Envelope 元素

必需的 SOAP 的 Envelope 元素是 SOAP 消息的根元素。它可把 XML 文档定义为 SOAP 消息。

实例

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

xmlns:soap 命名空间

SOAP 消息必须拥有与命名空间 "http://www.w3.org/2001/12/soap-envelope" 相关联的一个 Envelope 元素。

如果使用了不同的命名空间，应用程序会发生错误，并抛弃此消息。

encodingStyle 属性

SOAP 的 encodingStyle 属性用于定义在文档中使用的数据类型。此属性可出现在任何 SOAP 元素中，并会被应用到元素的内容及元素的所有子元素上。

SOAP 消息没有默认的编码方式。

语法

```
soap:encodingStyle="_URI_"
```

实例

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

SOAP Header 元素

可选的 SOAP Header 元素包含头部信息。

SOAP Header 元素

可选的 SOAP Header 元素可包含有关 SOAP 消息的应用程序专用信息（比如认证、支付等）。

如果 Header 元素被提供，则它必须是 Envelope 元素的第一个子元素。

注意： 所有 Header 元素的直接子元素必须是合格的命名空间。

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

上面的例子包含了一个带有一个 "Trans" 元素的头部，它的值是 234，此元素的 "mustUnderstand" 属性的值是 "1"。

SOAP 在默认的命名空间中 ("http://www.w3.org/2001/12/soap-envelope") 定义了三个属性。

这三个属性是：actor、 mustUnderstand 以及 encodingStyle。这些被定义在 SOAP 头部的属性可定义容器如何对 SOAP 消息进行处理。

mustUnderstand 属性

SOAP 的 mustUnderstand 属性可用于标识标题项对于要对其进行处理的接收者来说是强制的还是可选的。

假如您向 Header 元素的某个子元素添加了 "mustUnderstand="1"，则它可指示处理此头部的接收者必须认可此元素。假如此接收者无法认可此元素，则在处理此头部时必须失效。

语法

```
soap:mustUnderstand="0|1"
```

实例

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

actor 属性

通过沿着消息路径经过不同的端点，SOAP 消息可从某个发送者传播到某个接收者。并非 SOAP 消息的所有部分均打算传送到 SOAP 消息的最终端点，不过，另一个方面，也许打算传送给消息路径上的一个或多个端点。

SOAP 的 actor 属性可被用于将 Header 元素寻址到一个特定的端点。

语法

```
soap:actor="_URI_"
```

实例

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
  soap:actor="http://www.w3schools.com/appml/">234
  </m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

encodingStyle 属性

SOAP 的 encodingStyle 属性用于定义在文档中使用的数据类型。此属性可出现在任何 SOAP 元素中，并会被应用到元素的内容及元素的所有子元素上。

SOAP 消息没有默认的编码方式。

语法

```
soap:encodingStyle="_URI_"
```

SOAP Body 元素

强制使用的 SOAP Body 元素包含实际的 SOAP 消息。

SOAP Body 元素

必需的 SOAP Body 元素可包含打算传送到消息最终端点的实际 SOAP 消息。

SOAP Body 元素的直接子元素可以是合格的命名空间。

实例

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

上面的例子请求苹果的价格。请注意，上面的 m:GetPrice 和 Item 元素是应用程序专用的元素。它们并不是 SOAP 标准的一部分。

而一个 SOAP 响应应该类似这样：

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

SOAP Fault 元素

SOAP Fault 元素用于存留 SOAP 消息的错误和状态信息。

SOAP Fault 元素

可选的 SOAP Fault 元素用于指示错误消息。

如果已提供了 Fault 元素，则它必须是 Body 元素的子元素。在一条 SOAP 消息中，Fault 元素只能出现一次。

SOAP 的 Fault 元素拥有下列子元素：

子元素	描述
<faultcode>	供识别故障的代码
<faultstring>	可供人阅读的有关故障的说明
<faultactor>	有关是谁引发故障的信息
<detail>	存留涉及 Body 元素的应用程序专用错误信息

SOAP Fault 代码

在下面定义的 faultcode 值必须用于描述错误时的 faultcode 元素中：

错误	描述
VersionMismatch	SOAP Envelope 元素的无效命名空间被发现
MustUnderstand	Header 元素的一个直接子元素（带有设置为 "1" 的 mustUnderstand 属性）无法被理解。
Client	消息被不正确地构成，或包含了不正确的信息。
Server	服务器有问题，因此无法处理进行下去。

SOAP HTTP 协议

HTTP 协议

HTTP 在 TCP/IP 之上进行通信。HTTP 客户机使用 TCP 连接到 HTTP 服务器。在建立连接之后，客户机可向服务器发送 HTTP 请求消息：

```
POST /item HTTP/1.1
Host: 189.123.255.239
Content-Type: text/plain
Content-Length: 200
```

随后服务器会处理此请求，然后向客户机发送一个 HTTP 响应。此响应包含了可指示请求状态的状态代码：

```
200 OK
Content-Type: text/plain
Content-Length: 200
```

在上面的例子中，服务器返回了一个 200 的状态代码。这是 HTTP 的标准成功代码。

假如服务器无法对请求进行解码，它可能会返回类似这样的信息：

```
400 Bad Request
Content-Length: 0
```

SOAP HTTP Binding

SOAP 方法指的是遵守 SOAP 编码规则的 HTTP 请求/响应。

HTTP + XML = SOAP

SOAP 请求可能是 HTTP POST 或 HTTP GET 请求。

HTTP POST 请求规定至少两个 HTTP 头：Content-Type 和 Content-Length。

Content-Type

SOAP 的请求和响应的 Content-Type 头可定义消息的 MIME 类型，以及用于请求或响应的 XML 主体的字符编码（可选）。

语法

```
Content-Type: MIMEType; charset=character-encoding
```

实例

```
POST /item HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8
```

Content-Length

SOAP 的请求和响应的 Content-Length 头规定请求或响应主体的字节数。

语法

```
Content-Length: bytes
```

实例

```
POST /item HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 250
```

SOAP 实例

一个 SOAP 实例

在下面的例子中，一个 GetStockPrice 请求被发送到了服务器。此请求有一个 StockName 参数，而在响应中则会返回一个 Price 参数。此功能的命名空间被定义在此地址中："http://www.example.org/stock"

SOAP 请求：

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

SOAP 响应：

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

WSDL 教程

WSDL 简介

WSDL 是基于 XML 的用于描述 Web Services 以及如何访问 Web Services 的语言。

您应当具备的基础知识

在继续学习之前，您需要对下面的知识有基本的了解：

- XML
- XML 命名空间
- XML Schema

如果您希望首先学习这些项目，请访问我们的 XML 系列教程。

什么是 WSDL?

- WSDL 指网络服务描述语言
- WSDL 使用 XML 编写
- WSDL 是一种 XML 文档
- WSDL 用于描述网络服务
- WSDL 也可用于定位网络服务
- WSDL 还不是 W3C 标准

WSDL 可描述网络服务（Web Services）

WSDL 指网络服务描述语言 (Web Services Description Language)。

WSDL 是一种使用 XML 编写的文档。这种文档可描述某个 Web service。它可规定服务的位置，以及此服务提供的操作（或方法）。

在 W3C 的 WSDL 发展史

在 2001 年 3 月，WSDL 1.1 被 IBM、微软作为一个 W3C 记录（W3C note）提交到有关 XML 协议的 W3C XML 活动，用于描述网络服务。

（W3C 记录仅供讨论。一项 W3C 记录的发布并不代表它已被 W3C 或 W3C 团队亦或任何 W3C 成员认可。）

在 2002 年 7 月，W3C 发布了第一个 WSDL 1.2 工作草案。

请在我们的 W3C 教程 阅读更多有关规范的状态及时间线。

WSDL 文档

WSDL 文档仅仅是一个简单的 XML 文档。

它包含一系列描述某个 web service 的定义。

WSDL 文档结构

WSDL 文档是利用这些主要的元素来描述某个 web service 的：

元素	定义
<portType>	web service 执行的操作
<message>	web service 使用的消息
<types>	web service 使用的数据类型
<binding>	web service 使用的通信协议

一个 WSDL 文档的主要结构是类似这样的：

```
<definitions>

<types>
  data type definitions........
</types>

<message>
  definition of the data being communicated....
</message>

<portType>
  set of operations......
</portType>

<binding>
  protocol and data format specification....
</binding>

</definitions>
```

WSDL 文档可包含其它的元素，比如 extension 元素，以及一个 service 元素，此元素可把若干个 web services 的定义组合在一个单一的 WSDL 文档中。

h2>WSDL 端口

<portType> 元素是最重要的 WSDL 元素。

它可描述一个 web service、可被执行的操作，以及相关的消息。

可以把 <portType> 元素比作传统编程语言中的一个函数库（或一个模块、或一个类）。

WSDL 消息

<message> 元素定义一个操作的数据元素。

每个消息均由一个或多个部件组成。可以把这些部件比作传统编程语言中一个函数调用的参数。

WSDL types

<types> 元素定义 web service 使用的数据类型。

为了最大程度的平台中立性，WSDL 使用 XML Schema 语法来定义数据类型。

WSDL Bindings

<binding> 元素为每个端口定义消息格式和协议细节。

WSDL 实例

这是某个 WSDL 文档的简化的片段：

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

在这个例子中，*<portType>* 元素把 "glossaryTerms" 定义为某个端口的名称，把 "getTerm" 定义为某个操作的名称。

操作 "getTerm" 拥有一个名为 "getTermRequest" 的输入消息，以及一个名为 "getTermResponse" 的输出消息。

<message> 元素可定义每个消息的部件，以及相关联的数据类型。

对比传统的编程，glossaryTerms 是一个函数库，而 "getTerm" 是带有输入参数 "getTermRequest" 和返回参数 getTermResponse 的一个函数。

WSDL 端口

<portType> 元素是最重要的 WSDL 元素。

WSDL 端口

<*portType*> 元素是最重要的 WSDL 元素。

它可描述一个 web service、可被执行的操作，以及相关的消息。

可以把 <portType> 元素比作传统编程语言中的一个函数库（或一个模块、或一个类）。

操作类型

请求-响应是最普通的操作类型，不过 WSDL 定义了四种类型：

类型	定义
One-way	此操作可接受消息，但不会返回响应。
Request-response	此操作可接受一个请求并会返回一个响应
Solicit-response	此操作可发送一个请求，并会等待一个响应。
Notification	此操作可发送一条消息，但不会等待响应。

One-Way 操作

一个 one-way 操作的例子：

```
<message name="newTermValues">
  <part name="term" type="xs:string"/>
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
  </operation>
</portType >
```

在这个例子中，端口 "glossaryTerms" 定义了一个名为 "setTerm" 的 one-way 操作。

这个 "setTerm" 操作可接受新术语表项目消息的输入，这些消息使用一条名为 "newTermValues" 的消息，此消息带有输入参数 "term" 和 "value"。不过，没有为这个操作定义任何输出。

Request-Response 操作

一个 request-response 操作的例子：

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

在这个例子中，端口 "glossaryTerms" 定义了一个名为 "getTerm" 的 request-response 操作。

"getTerm" 操作会请求一个名为 "getTermRequest" 的输入消息，此消息带有一个名为 "term" 的参数，并将返回一个名为 "getTermResponse" 的输出消息，此消息带有一个名为 "value" 的参数。

WSDL 绑定

WSDL 绑定可为 web service 定义消息格式和协议细节。

绑定到 SOAP

一个 请求 - 响应 操作的例子：

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>

<binding type="glossaryTerms" name="b1">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
   <operation>
     <soap:operation soapAction="http://example.com/getTerm"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

binding 元素有两个属性 - name 属性和 type 属性。

name 属性定义 binding 的名称，而 type 属性指向用于 binding 的端口，在这个例子中是 "glossaryTerms" 端口。

soap:binding 元素有两个属性 - style 属性和 transport 属性。

style 属性可取值 "rpc" 或 "document"。在这个例子中我们使用 document。transport 属性定义了要使用的 SOAP 协议。在这个例子中我们使用 HTTP。

operation 元素定义了每个端口提供的操作符。

对于每个操作，相应的 SOAP 行为都需要被定义。同时您必须如何对输入和输出进行编码。在这个例子中我们使用了 "literal"。

WSDL UDDI

UDDI 是一种目录服务，企业可以使用它对 Web services 进行注册和搜索。 UDDI，英文为 "Universal Description, Discovery and Integration"，可译为"通用描述、发现与集成服务"。

什么是 UDDI？

UDDI 是一个独立于平台的框架，用于通过使用 Internet 来描述服务，发现企业，并对企业服务进行集成。

- UDDI 指的是通用描述、发现与集成服务
- UDDI 是一种用于存储有关 web services 的信息的目录。
- UDDI 是一种由 WSDL 描述的 web services 界面的目录。
- UDDI 经由 SOAP 进行通信
- UDDI 被构建入了微软的 .NET 平台

UDDI 基于什么？

UDDI 使用 W3C 和 IETF* 的因特网标准，比如 XML、HTTP 和 DNS 协议。

UDDI 使用 WSDL 来描述到达 web services 的界面

此外，通过采用 SOAP，还可以实现跨平台的编程特性，大家知道，SOAP 是 XML 的协议通信规范，可在 W3C 的网站找到相关的信息。

*注释：IETF - Internet Engineering Task Force

UDDI 的好处

任何规模的行业或企业都能得益于 UDDI。

在 UDDI 之前，还不存在一种 Internet 标准，可以供企业为它们的企业和伙伴提供有关其产品和服务的信息。也不存在一种方法，来集成到彼此的系统和进程中。

UDDI 规范帮助我们解决的问题：

- 使得在成百万当前在线的企业中发现正确的企业成为可能
- 定义一旦首选的企业被发现后如何启动商业
- 扩展新客户并增加对目前客户的访问
- 扩展销售并延伸市场范围
- 满足用户驱动的需要，为在全球 Internet 经济中快速合作的促进来清除障碍

UDDI 如何被使用

假如行业发布了一个用于航班比率检测和预订的 UDDI 标准，航空公司就可以把它们的服务注册到一个 UDDI 目录中。然后旅行社就能够搜索这个 UDDI 目录以找到航空公司预订界面。当此界面被找到后，旅行社就能够立即与此服务进行通信，这样由于它使用了一套定义良好的预订界面。

谁在支持 UDDI？

UDDI 是一个跨行业的研究项目，由所有主要的平台和软件提供商驱动，比如：Dell, Fujitsu, HP, Hitachi, IBM, Intel, Microsoft, Oracle, SAP, 以及 Sun, 它既是一个市场经营者的团体，也是一个电子商务的领导者。

已有数百家公司参与了这个 UDDI 团体。

RSS 教程

RSS 简介

您应当具备的基础知识

在继续学习之前，您需要对下面的知识有基本的了解：

- HTML / XHTML
- XML / XML 命名空间

如果您希望首先学习这些项目，请在我们的 首页 访问这些教程。

什么是 RSS？

- RSS 指 Really Simple Syndication（真正简易联合）
- RSS 使您有能力聚合（syndicate）网站的内容
- RSS 定义了非常简单的方法来共享和查看标题和内容
- RSS 文件可被自动更新
- RSS 允许为不同的网站进行视图的个性化
- RSS 使用 XML 编写

为什么使用 RSS？

RSS 被设计用来展示选定的数据。

如果没有 RSS，用户就不得不每日都来您的网站检查新的内容。对许多用户来说这样太费时了。通过 RSS feed（RSS 通常被称为 News feed 或 RSS feed），用户们可以使用 RSS 聚合器来更快地检查您的网站更新（RSS 聚合器是用来聚集并分类 RSS feed 的网站或软件）。

由于 RSS 数据很小巧并可快速加载，它可轻易地被类似移动电话或 PDA 的服务使用。

拥有相似内容的网站环（Web-rings）可以轻易地在它们的网站共享内容，使这些网站更出色更有价值。

谁应当使用 RSS？

那些极少更新内容的网管们不需要 RSS！

RSS 对那些频繁更新内容的网站是很有帮助的，比如：

- 新闻站点 - 列出新闻的标题、日期以及描述
- 企业 - 列出新闻和新产品
- 日程表 - 列出即将来临的安排和重要日期
- 站点更新 - 列出更新过的页面或新的页面

RSS 的未来

RSS 会无所不在！

成千上万的网站在使用 RSS，每天都有越来越多的人们认识到它的用处。

通过 RSS，因特网上的信息会更易查找，而网站开发者也可更容易地把他们的内容传播到特定的受众。

RSS的优势

选择您的新闻	通过**RSS**，你可以选择你想要查看新闻，感兴趣的新闻和与你的工作有关的信息。
删除不需要的信息	您可以用RSS（终于）单独从无用信息（垃圾邮件）获得想要的信息！
增加你的网站流量	通过RSS，你可以创建自己的新闻频道，并发布到互联网！

RSS 历史

RSS 已发布了很多不同的版本。

RSS 的历史

RSS 的历史

- **1997** 年 - Dave Winer 开发出 scriptingNews。RSS 由此诞生。
- **1999** 年 - Netscape 开发出 RSS 0.90 （由 scriptingNews 支持的）。这是带有一个 RDF header 的简单的 XML。
- **1999** 年- Dave Winer 在 UserLand 公司开发出 scriptingNews 2.0b1 （包含了 Netscape 的 RSS 0.90 的特定）。
- **1999** 年 - Netscape 开发出 RSS 0.91。在此版本中，他们删除了那个 RDF header，但是包含了大多数来自 scriptingNews 2.0b1 的特性。
- **1999** 年 - UserLand 摆脱了 scriptingNews，而仅仅使用 RSS 0.91。
- Netscape 停止了 RSS 的研发
- **2000** 年 - UserLand 发布了正式的 0.91 规范
- **2000** 年 - 一个由 Rael Dornfest 领导的团队在 O'Reilly 开发出 RSS 1.0。此格式使用了 RDF 和命名空间。此版本常被混淆为 0.91 的新版本，不过它是完全不依赖 RSS 0.91 的新格式。
- **2000** 年 - Dave Winer 在 UserLand 公司开发出 RSS 0.92。
- **2002** 年 - Dave Winer 在离开 Userland 之后开发出 RSS 2.0。
- **2003** 年 - 正式的 RSS 2.0 规范发布。

不同之处在哪里？

RSS 1.0 是唯一使用 W3C 的 RDF（资源描述框架） 标准进行开发的版本。

RDF 所蕴含的理念是帮助建立一张语义网。在此阅读更多有关 RDF 和语义网的内容。虽然这与普通用户的关系不大，但是通过使用 Web 标准，对于个人和应用程序来说数据交换会更加容易。

我应该使用哪个 RSS 版本？

RSS 0.91 和 RSS 2.0 较之 RSS 1.0 更容易理解。我们的教程基于 RSS 2.0。

存在 RSS 的 Web 标准吗？

没有正式的标准针对 RSS。

- 所有 RSS feeds 中大约 50% 使用 RSS 0.91。
- 大约 25% 使用 RSS 1.0。
- 最后的 25% 使用 RSS 0.9x 版本或 RSS 2.0。

RSS 语法

RSS 2.0 的语法很简单，也很严格。/p>

RSS 如何工作

RSS 用于在网站间分享信息。

使用 RSS，您在名为聚合器的公司注册您的内容。

步骤之一是，创建一个 RSS 文档，然后使用 .xml 后缀来保存它。然后把此文件上传到您的网站。接下来，通过一个 RSS 聚合器来注册。每天，聚合器都会到被注册的网站搜索 RSS 文档，校验其链接，并显示有关 feed 的信息，这样客户就能够链接到使他们产生兴趣的文档。

提示：请在 RSS 发布 这一节浏览免费的 RSS 聚合器服务。

RSS 实例

RSS 文档使用一种简单的自我描述的语法。

让我们看一个简单的 RSS 文档：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">

<channel>
  <title>W3Schools Home Page</title>
  <link>http://www.w3schools.com</link>
  <description>Free web building tutorials</description>
  <item>
    <title>RSS Tutorial</title>
    <link>http://www.w3schools.com/rss</link>
    <description>New RSS tutorial on W3Schools</description>
  </item>
  <item>
    <title>XML Tutorial</title>
    <link>http://www.w3schools.com/xml</link>
    <description>New XML tutorial on W3Schools</description>
  </item>
</channel>

</rss>
```

文档中的第一行：XML 声明 - 定义了文档中使用的 XML 版本和字符编码。此例子遵守 1.0 规范，并使用 ISO-8859-1 (Latin-1/West European) 字符集。

下一行是标识此文档是一个 RSS 文档的 RSS 声明（此例是 RSS version 2.0）。

下一行含有 <channel> 元素。此元素用于描述 RSS feed。

<channel> 元素有三个必需的子元素：

- <title> - 定义频道的标题。（比如 w3school 首页）
- <link> - 定义到达频道的超链接。（比如 www.w3school.com.cn）
- <description> - 描述此频道（比如免费的网站建设教程）

每个 <channel> 元素可拥有一个或多个 <item> 元素。

每个 <item> 元素可定义 RSS feed 中的一篇文章或 "story"。

<item> 元素拥有三个必需的子元素：

- <title> - 定义项目的标题。（比如 RSS 教程）
- <link> - 定义到达项目的超链接。（比如 http://www.w3school.com.cn/rss）
- <description> - 描述此项目（比如 w3school 的 RSS 教程）

最后，后面的两行关闭 <channel> 和 <rss> 元素。

RSS 中的注释

在 RSS 中书写注释的语法与 HTML 的语法类似：

```
<!-- This is an RSS comment -->
```

RSS 使用 XML 来编写

因为 RSS 也是 XML，请记住：

- 所有的元素必许拥有关闭标签
- 元素对大小写敏感
- 元素必需被正确地嵌套
- 属性值必须带引号

RSS <channel> 元素

RSS 的 <channel> 元素可描述 RSS feed。

RSS <channel> 元素

请看下面这个 RSS 文档：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">

<channel>
  <title>W3Schools Home Page</title>
  <link>http://www.w3schools.com</link>
  <description>Free web building tutorials</description>
  <item>
    <title>RSS Tutorial</title>
    <link>http://www.w3schools.com/rss</link>
    <description>New RSS tutorial on W3Schools</description>
  </item>
</channel>

</rss>
```

正如前面提到的，<channel> 元素可描述 RSS feed，而拥有三个必需的子元素：

- <title> - 定义频道的标题。（比如 w3school 首页）
- <link> - 定义到达频道的超链接。（比如 www.w3school.com.cn）
- <description> - 描述此频道（比如免费的网站建设教程）

<channel> 通常包含一个或多个 <item> 元素。每个 <item> 元素可定义 RSS feed 中的一篇文章或 "story"。

此外，还存在若干个可选的 <channel> 的子元素。我们会在后面讲解最重要的几个。

<category> 元素

<category> 子元素用于为 feed 规定种类。

<category> 子元素使 RSS 聚合器基于类别对网站进行分组成为可能。

上面的 RSS 文档的类别可能会是：

```
<category>Web development</category>
```

<copyright> 元素

<copyright> 子元素会告知有关版本资料的信息。

上面的 RSS 文档的版本可能会是

```
<copyright>2006 Refsnes Data as. All rights reserved.</copyright>
```

<image> 元素

<image> 子元素可在聚合器提供某个 feed 时显示一幅图像。

<image> 有三个必需的子元素：

- <url> - 定义引用图像的 URL
- <title> - 定义图像无法被显示时显示的文本
- <link> - 定义到达提供此频道的网站的超链接

上面的 RSS 文档的图像可能是这样的：

```
<image>
  <url>http://www.w3schools.com/images/logo.gif</url>
  <title>W3Schools.com</title>
  <link>http://www.w3schools.com</link>
</image>
```

<language> 元素

<language> 子元素用于规定用来编写文档的语言。

<language> 元素使 RSS 聚合器基于语言来对网站进行分组成为可能。

上面的 RSS 文档的语言可能是：

```
<language>en-us</language>
```

RSS <channel> 参考手册

元素	描述
<category>	可选的。为 feed 定义所属的一个或多个种类。
<cloud>	可选的。注册进程，以获得 feed 更新的立即通知。
<copyright>	可选。告知版权资料。
<description>	必需的。描述频道。
<docs>	可选的。规定指向当前 RSS 文件所用格式说明的 URL。
<generator>	可选的。规定用于生成 feed 的程序。
<image>	可选的。在聚合器呈现某个 feed 时，显示一个图像。
<language>	可选的。规定编写 feed 所用的语言。
<lastBuildDate>	可选的。定义 feed 内容的最后修改日期。
<link>	必需的。定义指向频道的超链接。
<managingEditor>	可选的。定义 feed 内容编辑的电子邮件地址。
<pubDate>	可选的。为 feed 的内容定义最后发布日期。
<rating>	可选的。feed 的 PICS 级别。
<skipDays>	可选的。规定忽略 feed 更新的天。
<skipHours>	可选的。规定忽略 feed 更新的小时。
<textInput>	可选的。规定应当与 feed 一同显示的文本输入域。
<title>	必需的。定义频道的标题。
<ttl>	可选的。指定从 feed 源更新此 feed 之前，feed 可被缓存的分钟数。
<webMaster>	可选的。定义此 feed 的 web 管理员的电子邮件地址。

RSS <item> 元素

每个 **<item>** 元素可定义 **RSS feed** 中的一篇文章或 **"story"**。

<item> 元素

请看下面的 RSS 文档：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">

<channel>
  <title>W3Schools Home Page</title>
  <link>http://www.w3schools.com</link>
  <description>Free web building tutorials</description>
  <item>
    <title>RSS Tutorial</title>
    <link>http://www.w3schools.com/rss</link>
    <description>New RSS tutorial on W3Schools</description>
  </item>
</channel>

</rss>
```

正如前面提到的，每个 <item> 元素可定义 RSS feed 中的一篇文章或 story"。

<item> 元素拥有三个必需的子元素：

- <title> - 定义项目的标题。（比如 RSS 教程）
- <link> - 定义到达项目的超链接。（比如 http://www.w3cschool.cc/rss）
- <description> - 描述此项目（比如 w3school 的 RSS 教程）

此外，存在若干个 <item> 的可选的子元素，我们会在下面介绍最重要的几个。

<author> 元素

<author> 子元素用于规定一个项目的作者的电子邮件地址。

注释：为了防止垃圾邮件，一些开发者不会使用这个 <author> 元素。

上面的 RSS 文档中项目的作者可能是：

```
<author>hege@refsnesdata.no</author>
```

<comments> 元素

<comments> 子元素允许把一个项目连接到有关此项目的注释。

上面的 RSS 文档中项目的注释可能这样的：

```
<comments>http://www.w3schools.com/comments</comments>
```

<enclosure> 元素

<enclosure> 子元素允许将一个媒体文件导入一个项中。

<enclosure> 元素有三个必需的属性：

- url - 定义指向此媒体文件的 URL
- length - 定义此媒体文件的长度（字节）
- type - 定义媒体文件的类型

在上面的 RSS 文档中，被包含在项目中的媒体文件可能是这样的：

```
<enclosure url="http://www.w3schools.com/rss/rss.mp3"
length="5000" type="audio/mpeg" />
```

RSS <item> 参考手册

元素	描述
<author>	可选的。规定项目作者的电子邮件地址。
<category>	可选的。定义项目所属的一个或多个类别。
<comments>	可选的。允许项目连接到有关此项目的注释（文件）。
<description>	必需的。描述此项目。
<enclosure>	可选的。允许将一个媒体文件导入一个项中。
<guid>	可选的。为项目定义一个唯一的标识符。
<link>	必需的。定义指向此项目的超链接。
<pubDate>	可选的。定义此项目的最后发布日期。
<source>	可选的。为此项目指定一个第三方来源。
<title>	必需的。定义此项目的标题。

RSS 发布您的 Feed

只能当其他人能够找到您的 RSS 文档时，它才是有用的。

把您的 RSS 发布到 Web 上

现在是时候把您的 RSS 文件上传到网上了。下面是具体的步骤：

1.为您的 RSS 命名。请注意文件必须有 .xml 的后缀。

2. 验证您的 RSS 文件。（可以在 http://www.feedvalidator.org 找到很好的验证器）。

3. 把 RSS 文件上传到您的 web 服务器上的 web 目录。

4. 把这个小的橙色按钮 或 拷贝到您的 web 目录。

5. 在你希望向外界提供 RSS 的页面上放置这个小按钮。然后向这个按钮添加一个指向 RSS 文件的链接。代码应该类似这样：

```
<a href="http://www.w3cschools.cc/rss/myfirstrss.xml">
<img src="http://www.w3cschools.cc/rss/rss.gif" width="36" height="14">
</a>.
```

6. 把你的 RSS feed 提交到 RSS Feed 目录。要注意！feed 的 URL 不是你的页面，而是您的指向您的 feed 的 URL，比如 "http://www.w3cschools.cc/rss/myfirstrss.xml"。 此处提供一些免费的 RSS 聚合服务：

- Syndic8: Register here
- Newsisfree: Register here

7. 在重要的搜索引擎注册您的 feed ：

- Yahoo - http://publisher.yahoo.com/promote.php
- Google - http://www.google.com/intl/zh-cn/webmasters/addfeed.html
- MSN - http://rss.msn.com/publisher.armx

8. 更新您的 feed - 现在您已获得了来自 Google、Yahoo、以及 MSN 的 RSS feed 按钮。请您务必经常更新您的内容，并保持 RSS feed 的长期可用。

我可以自己来维护 RSS feed 吗？

确保 RSS feed 按照您期望的方式工作的最好的办法，就是自己来维护它。

不过，这么做很费时，特别是对于大量的更新工作来讲。

替代方案是使用一个第三方的自动 RSS。

自动的 RSS

如果您不想自己去更新 RSS feed，有一些工具和服务可以为您自动地完成工作，比如：

- MyRSSCreator - 在 10 分钟之内提供自动的、可靠的 RSS 服务
- FeedFire - 提供免费的 RSS feed 创建和分发

对于那些仅需要一个用于个人网站的 RSS feed 的用户来说，一些流行的 blog (Web Log) 管理器可提供内建的 RSS 服务：

- WordPress
- Blogger
- Radio

RSS 阅读器

RSS 阅读器用于读取 RSS feed！

RSS 阅读器可供许多不同的设备和操作系统使用。

RSS 阅读器

有很多不同的 RSS 阅读器。某些以 web services 的形式来工作，而某些则运行于 windows（或 Mac、PDA 或 UNIX）。

这是一些我尝试过并钟爱的阅读器：

RSS 阅读器

有很多不同的 RSS 阅读器。某些以 web services 的形式来工作，而某些则运行于 windows（或 Mac、PDA 或 UNIX）。

这是一些我尝试过并钟爱的阅读器：

RSS 阅读器

有很多不同的 RSS 阅读器。某些以 web services 的形式来工作，而某些则运行于 windows（或 Mac、PDA 或 UNIX）。

这是一些我尝试过并钟爱的阅读器：

- NewsGator Online 一个免费的在线 RSS 阅读器。包含 Outlook 同步，通过 Media Center Edition 查看电视内容，以及 blog 和标题的发布。
- RssReader 基于 Windows 的免费 RSS 阅读器。支持 RSS versions 0.9x、1.0 以及 2.0 和 Atom 0.1, 0.2 以及 0.3。
- FeedDemon 基于 Windows 的 RSS 阅读器。使用很简便，界面很有条理。可以免费下载！
- blogbot 一个针对 Outlook 或 Internet Explorer 的 RSS 阅读器插件。针对 Internet Explorer 的简化版是免费的。

提示：Mozilla Firefox 浏览器拥有内建的 RSS 阅读器。在您访问提供 RSS feed 的网站时，会在地址栏看到 Firefox 的 RSS 图标。点击这个图标可查看一个不同 feed 的列表，在此可选择你需要阅读的 feed。

我已经有一个 RSS 阅读器了，接下来怎么做呢？

点击您希望阅读的 RSS feed 旁边的橙色小图标 RSS 或 XML，把浏览器窗口的 URL 拷贝粘贴到您的 RSS 阅读器即可。

RSS 参考手册

RSS <channel> 元素

请点击元素列中的具体元素，获得更详细的信息。

元素	描述
<category>	可选的。为 feed 定义所属的一个或多个种类。
<cloud>	可选的。注册进程，以获得 feed 更新的立即通知。
<copyright>	可选。告知版权资料。
<description>	必需的。描述频道。
<docs>	可选的。规定指向当前 RSS 文件所用格式说明的 URL。
<generator>	可选的。规定用于生成 feed 的程序。
<image>	可选的。在聚合器呈现某个 feed 时，显示一个图像。
<language>	可选的。规定编写 feed 所用的语言。
<lastBuildDate>	可选的。定义 feed 内容的最后修改日期。
<link>	必需的。定义指向频道的超链接。
<managingEditor>	可选的。定义 feed 内容编辑的电子邮件地址。
<pubDate>	可选的。为 feed 的内容定义最后发布日期。
<rating>	可选的。feed 的 PICS 级别。
<skipDays>	可选的。规定忽略 feed 更新的天。
<skipHours>	可选的。规定忽略 feed 更新的小时。
<textInput>	可选的。规定应当与 feed 一同显示的文本输入域。
<title>	必需的。定义频道的标题。
<ttl>	可选的。指定从 feed 源更新此 feed 之前，feed 可被缓存的分钟数。
<webMaster>	可选的。定义此 feed 的 web 管理员的电子邮件地址。

RSS <item> 元素

元素	描述
<author>	可选的。规定项目作者的电子邮件地址。
<category>	可选的。定义项目所属的一个或多个类别。
<comments>	可选的。允许项目连接到有关此项目的注释（文件）。
<description>	必需的。描述此项目。
<enclosure>	可选的。允许将一个媒体文件导入一个项中。
<guid>	可选的。为项目定义一个唯一的标识符。
<link>	必需的。定义指向此项目的超链接。
<pubDate>	可选的。定义此项目的最后发布日期。
<source>	可选的。为此项目指定一个第三方来源。
<title>	必需的。定义此项目的标题。

RDF 教程

RDF 简介

资源描述框架（RDF）是用于描述网络资源的 W3C 标准，比如网页的标题、作者、修改日期、内容以及版权信息。

您应当具备的基础知识

在继续学习之前，您需要对下面的知识有基本的了解：

- HTML
- XHTML
- XML
- XML 命名空间

如果您希望首先学习这些项目，请访问我们的 首页。

什么是 RDF？

- RDF 指资源描述框架（Resource Description Framework）
- RDF 是一个用于描述 Web 上的资源的框架
- RDF 提供了针对数据的模型以及语法，这样独立的团体们就可以交换和使用它
- RDF 被设计为可被计算机阅读和理解
- RDF 被设计的目的不是为了向人们显示出来
- RDF 使用 XML 编写
- RDF 是 W3C 语义网络活动的组成部分
- RDF 是一个 W3C 推荐标准

RDF - 应用举例

- 描述购物项目的属性，比如价格以及可用性
- 描述 Web 事件的时间表
- 描述有关网页的信息，比如内容、作者以及被创建和修改的日期
- 描述网络图片的内容和等级
- 描述针对搜索引擎的内容
- 描述电子图书馆

RDF 被设计为可被计算机读取

RDF 被设计为提供一种描述信息的通用方法，这样就可以被计算机应用程序读取并理解。

RDF 描述不是被设计用来在网络上显示的。

RDF 使用 XML 编写

RDF 文档使用 XML 编写。被 RDF 使用的 XML 语言被称为 RDF/XML。

通过使用 XML，RDF 信息可以轻易地在使用不同类型的操作系统和应用语言的计算机之间进行交换。

RDF 和语义网

RDF 语言是 W3C 的语义网活动的组成部分。W3C 的"语义网远景 (Semantic Web Vision)"的目标是：

- Web 信息拥有确切的含义
- Web 信息可被计算机理解并处理
- 计算机可从 Web 上整合信息

RDF 是 W3C 标准

RDF 在 2004 年 2 月成为 W3C 标准。

W3C 推荐（标准）被业界以及 web 团体奉为 web 标准。W3C推荐标准 是由 W3C 工作组开发并经 W3C 成员评审的稳定规范。

可以通过此链接找到官方的 W3C 推荐标准。

http://www.w3.org/RDF/

RDF 规则

RDF 使用 Web 标识符 (URIs) 来标识资源。

RDF 使用属性和属性值来描述资源。

RDF 资源、属性和属性值

RDF 使用 Web 标识符来标识事物，并通过属性和属性值来描述资源。

对资源、属性和属性值的解释：

- 资源是可拥有 URI 的任何事物，比如 "http://www.w3cschool.cc/rdf"
- 属性是拥有名称的资源，比如 "author" 或 "homepage"
- 属性值是某个属性的值，比如 "David" 或 "http://www.w3cschool.cc" （请注意一个属性值可以是另外一个资源）

下面的 RDF 文档可描述资源 "http://www.w3cschool.cc/rdf"：

```
<?xml version="1.0"?>

<RDF>
  <Description about="http://www.w3cschool.cc/rdf">
    <author>Jan Egil Refsnes</author>
    <homepage>http://www.w3cschool.cc</homepage>
  </Description>
</RDF>
```

 上面是一个简化的例子。命名空间被忽略了。

RDF 陈述

资源、属性和属性值的组合可形成一个陈述（被称为陈述的主体、谓语和客体）。

请看一些陈述的具体例子，来加深理解：

陈述："The author of http://www.w3cschool.cc/rdf is David."

- 陈述的主体是：http://www.w3cschool.cc/rdf
- 谓语是：author
- 客体是：David

陈述："The homepage of http://www.w3cschool.cc/rdf is http://www.w3cschool.cc".

- 陈述的主体是：http://www.w3cschool.cc/rdf

- 谓语是：homepage
- 客体是：http://www.w3cschool.cc

RDF 实例

RDF 实例

这是一个 CD 列表的其中几行:

标题	艺术家	国家	公司	价格	年份
Empire Burlesque	Bob Dylan	USA	Columbia	10.90	1985
Hide your heart	Bonnie Tyler	UK	CBS Records	9.90	1988

这是一个 RDF 文档的其中几行：

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
  <cd:artist>Bob Dylan</cd:artist>
  <cd:country>USA</cd:country>
  <cd:company>Columbia</cd:company>
  <cd:price>10.90</cd:price>
  <cd:year>1985</cd:year>
</rdf:Description>

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Hide your heart">
  <cd:artist>Bonnie Tyler</cd:artist>
  <cd:country>UK</cd:country>
  <cd:company>CBS Records</cd:company>
  <cd:price>9.90</cd:price>
  <cd:year>1988</cd:year>
</rdf:Description>
.
.
.
</rdf:RDF>
```

此 RDF 文档的第一行是 XML 声明。这个 XML 声明之后是 RDF 文档的根元素：*<rdf:RDF>*。

xmlns:rdf 命名空间，规定了带有前缀 rdf 的元素来自命名空间 "http://www.w3.org/1999/02/22-rdf-syntax-ns#"。

xmlns:cd 命名空间，规定了带有前缀 cd 的元素来自命名空间 "http://www.recshop.fake/cd#"。

<rdf:Description> 元素包含了对被 *rdf:about* 属性标识的资源的描述。

元素：*<cd:artist>*、*<cd:country>*、*<cd:company>* 等是此资源的属性。

RDF 在线验证器

W3C 的 RDF 验证服务在您学习 RDF 时是很有帮助的。在此您可对 RDF 文件进行试验。

RDF 在线验证器可解析您的 RDF 文档，检查其中的语法，并为您的 RDF 文档生成表格和图形视图。

把下面这个例子拷贝粘贴到 W3C 的 RDF 验证器：

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:si="http://www.w3schools.com/rdf/">
<rdf:Description rdf:about="http://www.w3schools.com">
  <si:title>W3Schools.com</si:title>
  <si:author>Jan Egil Refsnes</si:author>
</rdf:Description>
</rdf:RDF>
```

在您对上面的例子进行解析后，结果将是类似这样的。

RDF 主要 元素

RDF 的主要元素是 **<RDF>** 以及可表示某个资源的 **<Description>** 元素。

<rdf:RDF> 元素

<rdf:RDF> 是 RDF 文档的根元素。它把 XML 文档定义为一个 RDF 文档。它也包含了对 RDF 命名空间的引用：

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  ..._Description goes here..._
</rdf:RDF>
```

<rdf:Description> 元素

<rdf:Description> 元素可通过 about 属性标识一个资源。

<rdf:Description> 元素可包含描述资源的那些元素：

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
  <cd:artist>Bob Dylan</cd:artist>
  <cd:country>USA</cd:country>
  <cd:company>Columbia</cd:company>
  <cd:price>10.90</cd:price>
  <cd:year>1985</cd:year>
</rdf:Description>

</rdf:RDF>
```

artist、country、company、price 以及 year 这些元素被定义在命名空间 http://www.recshop.fake/cd# 中。此命名空间在 RDF 之外（并非 RDF 的组成部分）。RDF 仅仅定义了这个框架。而 artist、country、company、price 以及 year 这些元素必须被其他人（公司、组织或个人等）进行定义。

属性（property）来定义属性（attribute）

属性元素（property elements）也可作为属性（attributes）来被定义（取代元素）：

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Empire Burlesque"
cd:artist="Bob Dylan" cd:country="USA"
cd:company="Columbia" cd:price="10.90"
cd:year="1985" />

</rdf:RDF>
```

属性（property）来定义属性（attribute）

属性元素（property elements）也可作为属性（attributes）来被定义（取代元素）：

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
  <cd:artist rdf:resource="http://www.recshop.fake/cd/dylan" />
  ...
  ...
</rdf:Description>

</rdf:RDF>
```

上面的例子中，属性 artist 没有值，但是却引用了一个对包含有关艺术家的信息的资源。

RDF 容器 Elements

RDF 容器用于描述一组事物。举个例子，把某本书的作者列在一起。

下面的 **RDF** 元素用于描述这些的组：**<Bag>**、**<Seq>** 以及 **<Alt>**。

h2> 元素

<rdf:Bag> 元素用于描述一个规定为无序的值的列表。

<rdf:Bag> 元素可包含重复的值。

实例

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Beatles">
  <cd:artist>
    <rdf:Bag>
      <rdf:li>John</rdf:li>
      <rdf:li>Paul</rdf:li>
      <rdf:li>George</rdf:li>
      <rdf:li>Ringo</rdf:li>
    </rdf:Bag>
  </cd:artist>
</rdf:Description>

</rdf:RDF>
```

<rdf:Seq> 元素

<rdf:Seq> 元素用于描述一个规定为有序的值的列表（比如一个字母顺序的排序）。

<rdf:Bag> 元素可包含重复的值。

实例

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Beatles">
  <cd:artist>
    <rdf:Seq>
      <rdf:li>George</rdf:li>
      <rdf:li>John</rdf:li>
      <rdf:li>Paul</rdf:li>
      <rdf:li>Ringo</rdf:li>
    </rdf:Seq>
  </cd:artist>
</rdf:Description>

</rdf:RDF>
```

<rdf:Alt> 元素

<rdf:Alt> 元素用于一个可替换的值的列表（用户仅可选择这些值的其中之一）。

实例

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Descriptio
rdf:about="http://www.recshop.fake/cd/Beatles">
  <cd:format>
    <rdf:Alt>
      <rdf:li>CD</rdf:li>
      <rdf:li>Record</rdf:li>
      <rdf:li>Tape</rdf:li>
    </rdf:Alt>
  </cd:format>
</rdf:Descriptio>

</rdf:RDF>
```

RDF 术语

在上面的例子中，我们在描述容器元素时已经讨论了"值的列表"。在 RDF 中，这些"值的列表"被称为成员（members）。

因此，我们可以这么说：

- 一个容器是一个包含事物的资源
- 被包含的事物被称为成员（不能称为"值的列表"）。

RDF 集合

RDF 集合用于描述仅包含指定成员的组。

rdf:parseType="Collection" 属性

正如在前面的章节所看到的，我们无法关闭一个容器。容器规定了所包含的资源为成员 - 它没有规定其他的成员是不被允许的。

RDF 集合用于描述仅包含指定成员的组。

集合是通过属性 rdf:parseType="Collection" 来描述的。

实例

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://recshop.fake/cd#">

<rdf:Description
rdf:about="http://recshop.fake/cd/Beatles">
  <cd:artist rdf:parseType="Collection">
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/George"/>
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/John"/>
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/Paul"/>
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/Ringo"/>
  </cd:artist>
</rdf:Description>

</rdf:RDF>
```

RDF Schema (RDFS)

RDF Schema (RDFS) 是对 RDF 的一种扩展。

RDF Schema 和 应用程序的类

RDF 通过类、属性和值来描述资源。

此外，RDF 还需要一种定义应用程序专业的类和属性的方法。应用程序专用的类和属性必须使用对 RDF 的扩展来定义。

RDF Schema 就是这样一种扩展。

RDF Schema (RDFS)

RDF Schema 不提供实际的应用程序专用的类和属性，而是提供了描述应用程序专用的类和属性的框架。

RDF Schema 中的类与面向对象编程语言中的类非常相似。这就使得资源能够作为类的实例和类的子类来被定义。

RDFS 实例

下面的实例演示了 RDFS 的能力的某些方面：

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.animals.fake/animals#">

<rdf:Description rdf:ID="animal">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="horse">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#animal"/>
</rdf:Description>

</rdf:RDF>
```

在上面的例子中，资源 "horse" 是类 "animal" 的子类。

简写的例子

由于一个 RDFS 类就是一个 RDF 资源，我们可以通过使用 rdfs:Class 取代 rdf:Description，并去掉 rdf:type 信息，来把上面的例子简写一下：

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.animals.fake/animals#">

<rdfs:Class rdf:ID="animal" />

<rdfs:Class rdf:ID="horse">
  <rdfs:subClassOf rdf:resource="#animal"/>
</rdfs:Class>

</rdf:RDF>
```

就是这样！

RDF 都柏林核心元数据倡议

都柏林核心元数据倡议 (DCMI) 已创建了一些供描述文档的预定义属性。

Dublin 核心

RDF 是元数据（关于数据的数据）。RDF 被用于描述信息资源。

都柏林核心是一套供描述文档的预定义属性。

第一份都柏林核心属性是于1995年 在俄亥俄州的都柏林的元数据工作组被定义的，目前由都柏林元数据倡议来维护。

属性	定义
Contributor	一个负责为资源内容作出贡献的实体(如作者)。
Coverage	资源内容的氛围或作用域
Creator	一个主要负责创建资源内容的实体。
Format	物理或数字的资源表现形式。
Date	在资源生命周期中某事件的日期。
Description	对资源内容的说明。
Identifier	一个对在给定上下文中的资源的明确引用
Language	资源智力内容所用的语言。
Publisher	一个负责使得资源内容可用的实体
Relation	一个对某个相关资源的引用
Rights	有关保留在资源之内和之上的权利的信息
Source	一个对作为目前资源的来源的资源引用。
Subject	一个资源内容的主题
Title	一个给资源起的名称
Type	资源内容的种类或类型。

通过浏览上面这个表格，我们可以发现 RDF 是非常适合表示都柏林核心信息的。

RDF 实例

下面的例子演示了都柏林核心属性在一个 RDF 文档中的使用：

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc= "http://purl.org/dc/elements/1.1/">

<rdf:Description rdf:about="http://www.w3schools.com">
  <dc:description>W3Schools - Free tutorials</dc:description>
  <dc:publisher>Refsnes Data as</dc:publisher>
  <dc:date>2008-09-01</dc:date>
  <dc:type>Web Development</dc:type>
  <dc:format>text/html</dc:format>
  <dc:language>en</dc:language>
</rdf:Description>

</rdf:RDF>
```

OWL 简介

OWL 是一门供处理 web 信息的语言。

在学习之前应具备的基础知识

在您学习 OWL 之前，应当对 XML、XML 命名空间以及 RDF 有基本的了解。

如果首先学习这些项目，请访问：

W3Cschool 的 XML 教程 和 RDF 教程。

什么是 OWL？

- OWL 指的是 web 本体语言
- OWL 构建在 RDF 的顶端之上
- OWL 用于处理 web 上的信息
- OWL 被设计为供计算机进行解释
- OWL 不是被设计为供人类进行阅读的
- OWL 由 XML 来编写
- OWL 拥有三种子语言
- OWL 是一项 web 标准

什么是本体？

本体"这个术语来自于哲学，它是研究世界上的各种实体以及他们是怎么关联的科学。

对于 web，本体则关于对 web 信息及 web 信息之间的关系的精确描述。

为什么使用 OWL?

OWL 是"语义网远景"的组成部分 - 目标是：

- Web 信息拥有确切的含义
- Web 信息可被计算机理解并处理
- 计算机可从 Web 上整合信息

OWL 被设计为供计算机来处理信息

OWL 被设计为提供一种通用的处理 Web 信息的内容的方法（而不是把它显示出来）。

OWL 被设计为由计算机应用程序来读取（而不是被人类）。

OWL 与 RDF 不同

OWL 与 RDF 有很多相似之处，但是较之 RDF， OWL 是一门具有更强机器解释能力的更强大的语言。

与 RDF 相比，OWL 拥有更大的词汇表以及更强大的语言。

OWL 子语言

OWL 有三门子语言：

- OWL Lite
- OWL DL (包含 OWL Lite)
- OWL Full (包含 OWL DL)

OWL 使用 XML 编写

通过使用 XML，OWL 信息可在使用不同类型的操作系统和应用语言的不同类型的计算机之间进行交换。

OWL 是一个 Web 标准

OWL 于 2004 年 2 月成为一项 W3C 的推荐标准。

W3C 推荐（标准）被业界以及 web 团体奉为 web 标准。W3C推荐标准 是由 W3C 工作组开发并经 W3C 成员评审的稳定规范。

在 w3c 有关 OWL 的文档：http://www.w3.org/2004/OWL/

RDF 参考手册

RDF 命名空间

RDF 命名空间(xmlns:rdf):: http://www.w3.org/1999/02/22-rdf-syntax-ns#

RDF 命名空间(xmlns:rdfs):: http://www.w3.org/2000/01/rdf-schema#

RDF 扩展名和MIME 类型

RDF 文件的推荐扩展名为**.rdf**,然而，扩展名**.XML**是经常被用来兼容旧的XML解析器。

MIME 类型：**"application/rdf+xml"**。

RDFS / RDF 类

元素	类	子类
rdfs:Class	All classes	
rdfs:Datatype	Data types	Class
rdfs:Resource	All resources	Class
rdfs:Container	Containers	Resource
rdfs:Literal	Literal values (text and numbers)	Resource
rdf:List	Lists	Resource
rdf:Property	Properties	Resource
rdf:Statement	Statements	Resource
rdf:Alt	Containers of alternatives	Container
rdf:Bag	Unordered containers	Container
rdf:Seq	Ordered containers	Container
rdfs:ContainerMembershipProperty	Container membership properties	Property
rdf:XMLLiteral	XML literal values	Literal

RDFS / RDF 属性

元素	领域	范围	描述
rdfs:domain	Property	Class	资源域
rdfs:range	Property	Class	资源的范围
rdfs:subPropertyOf	Property	Property	该属性是一个属性的子属性
rdfs:subClassOf	Class	Class	资源是一个类的子类
rdfs:comment	Resource	Literal	人类可读的资源描述
rdfs:label	Resource	Literal	人类可读的资源标签（名称）
rdfs:isDefinedBy	Resource	Resource	资源的定义
rdfs:seeAlso	Resource	Resource	关于资源的其他信息
rdfs:member	Resource	Resource	资源的成员
rdf:first	List	Resource	
rdf:rest	List	List	
rdf:subject	Statement	Resource	一个RDF陈述的资源主体
rdf:predicate	Statement	Resource	在一个RDF陈述的资源的谓词
rdf:object	Statement	Resource	一个RDF陈述的资源客体
rdf:value	Resource	Resource	value属性
rdf:type	Resource	Class	资源是一个类的实例

RDF 属性

元素	领域	范围	描述
rdf:about	定义所描述的资源		
rdf:Description	资源描述的容器		
rdf:resource	定义资源，以确定一个属性		
rdf:datatype	定义一个元素的数据类型		
rdf:ID	定义元素的ID		
rdf:li	定义列表		
rdf:_*n*	定义一个节点		
rdf:nodeID	定义元素节点的ID		
rdf:parseType	定义元素应如何解析		
rdf:RDF	一个RDF文档的根		
xml:base	定义了XML基础		
xml:lang	定义元素内容的语言		
rdf:aboutEach	（删除）		
rdf:aboutEachPrefix	（删除）		
rdf:bagID	（删除）		

描述为"删除" 的为最近从RDF标准删除元素。

XSL-FO 教程

XSL-FO 简介

XSL-FO 用于格式化供输出的 XML 数据。

学习之前应当具备的基础知识

在您学习 XSL-FO 之前，应当对 XML 和 XML 命名空间有基本的了解。

如果您希望首先学习这些项目，请阅读我们的 XML 教程。

什么是 XSL-FO？

- XSL-FO 是用于格式化 XML 数据的语言
- XSL-FO 指可扩展样式表语言格式化对象（Extensible Stylesheet Language Formatting Objects）
- XSL-FO 是基于 XML
- XSL-FO 是一个 W3C 推荐标准
- XSL-FO 目前通常被称为 XSL

XSL-FO 与格式化有关

XSL-FO 是一种基于 XML 的标记语言，用于描述向屏幕、纸或者其他媒介输出 XML 数据的格式化（信息）。

XSL-FO 通常被称为 XSL

为什么会存在这样的混淆呢？XSL-FO 和 XSL 是一回事吗？

可以这么说，不过我们需要向您作一个解释：

样式化（Styling）是关于转换信息和格式化信息两方面。在万维网联盟（W3C）编写他们的首个 XSL 工作草案的时候，这个草案包括了有关转换和格式化 XML 文档的语言语法。

后来，W3C 工作组把这个原始的草案分为独立的标准：

- XSLT，用于转换 XML 文档的语言
- XSL 或 XSL-FO，用于格式化 XML 文档的语言
- XPath，是通过元素和属性在 XML 文档中进行导航的语言

本教程的其余内容均与格式化 XML 文档有关：XSL-FO，也被称为 XSL。

XSL-FO 是一个 Web 标准

XSL-FO 在 2001 年 10 月 15 日 被确立为 W3C 推荐标准。通常被称为 XSL。

如需阅读更多有关 W3C 的 XSL 活动的内容，请阅读我们的 W3C 教程。

XSL-FO 文档

XSL-FO 文档

XSL-FO 文档是带有输出信息的 XML 文件。

XSL-FO 文档存储在以 .fo 或 .fob 为文件扩展名的文件中。您也可以把 XSL-FO 文档存储为以 .xml 为扩展名的文件，这样做的话可以使 XSL-FO 文档更易被 XML 编辑器存取。

XSL-FO 文档结构

XSL-FO 的文档结构如下所示：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

<fo:layout-master-set>
<fo:simple-page-master master-name="A4">
<!-- Page template goes here -->
</fo:simple-page-master>
</fo:layout-master-set>

<fo:page-sequence master-reference="A4">
<!-- Page content goes here -->
</fo:page-sequence>

</fo:root>
```

结构解释

XSL-FO 文档属于 XML 文档，因此也需要以 XML 声明来起始：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

<fo:root> 元素是 XSL-FO 文档的根元素。这个根元素也要声明 XSL-FO 的命名空间：

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
<!-- The full XSL-FO document goes here -->
</fo:root>
```

<fo:layout-master-set> 元素包含一个或多个页面模板：

```
<fo:layout-master-set>
<!-- All page templates go here -->
</fo:layout-master-set>
```

每个 <fo:simple-page-master> 元素包含一个单一的页面模板。每个模板必须有一个唯一的名称（master-name）：

```
<fo:simple-page-master master-name="A4">
<!-- One page template goes here -->
</fo:simple-page-master>
```

一个或多个 <fo:page-sequence> 元素可描述页面内容。master-reference 属性使用相同的名称来引用 simple-page-master 模板：

```
<fo:page-sequence master-reference="A4">
<!-- Page content goes here -->
</fo:page-sequence>
```

注释：master-reference 的值 "A4" 实际上并没有描述某个预定义的页面格式。它仅仅是一个名称。您可以使用任何名称，比如 "MyPage"、"MyTemplate" 等等。

XSL-FO 区域

XSL-FO 使用矩形框（区域）来显示输出。

XSL-FO 区域

XSL 格式化模型定义了一系列的矩形区域（框）来显示输出。

所有的输出（文本、图片，等等）都会被格式化到这些框中，然后会被显示或打印到某个目标媒介。

让我们研究一下下面这些区域：

- Pages（页面）
- Regions（区）
- Block areas（块区域）
- Line areas（行区域）
- Inline areas（行内区域）

XSL-FO Pages（页面）

XSL-FO 输出会被格式化到页面中。打印输出通常会分为许多分割的页面。浏览器输出经常会成为一个长的页面。

XSL-FO Pages（页面）包含区（Region）。

XSL-FO Regions（区）

每个 XSL-FO 页面均包含一系列的 Regions（区）：

- region-body（页面的主体）
- region-before（页面的页眉）
- region-after（页面的页脚）
- region-start（左侧栏）
- region-end（右侧栏）

XSL-FO Regions（区）包含块区域（Block Area）。

XSL-FO Block Areas（块区域）

XSL-FO Block Areas（块区域）定义了小的块元素（通常由一个新行开始），比如段落、表格以及列表。

XSL-FO Block Areas（块区域）包含其他的块区域，不过大多数时候它们包含的是行区域（Line Area）。

XSL-FO Line Areas（行区域）

XSL-FO Line Areas（行区域）定义了块区域内部的文本行。

XSL-FO Line Areas（行区域）包含行内区域（Inline Area）。

XSL-FO Inline Areas（行内区域）

XSL-FO Inline Areas（行内区域）定义了行内部的文本（着重号、单字符、图像，等等）。

XSL-FO 输出

XSL-FO 在 <fo:flow> 元素内部定义输出。

XSL-FO 页面（Page）、流（Flow）以及块（Block）

内容"块"会"流"入"页面"中，然后输出到媒介。

XSL-FO 输出通常被嵌套在 <fo:block> 元素内，<fo:block> 嵌套于 <fo:flow> 元素内，<fo:flow> 嵌套于 <fo:page-sequence> 元素内：

```
<fo:page-sequence>
<fo:flow flow-name="xsl-region-body">
<fo:block>
<!-- Output goes here -->
</fo:block>
</fo:flow>
</fo:page-sequence>
```

XSL-FO 实例

现在让我们看一个真实的 XSL-FO 实例：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

<fo:layout-master-set>
<fo:simple-page-master master-name="A4">
<fo:region-body />
</fo:simple-page-master>
</fo:layout-master-set>

<fo:page-sequence master-reference="A4">
<fo:flow flow-name="xsl-region-body">
<fo:block>Hello W3CSchool</fo:block>
</fo:flow>
</fo:page-sequence>

</fo:root>
```

以上代码的输出如下所示：

```
Hello W3CSchool
```

XSL-FO 流

XSL-FO 页面使用来自 <fo:flow> 元素的数据进行填充。

XSL-FO 页面序列（Page Sequences）

XSL-FO 使用 <fo:page-sequence> 元素来定义输出页面。

每个输出页面都会引用一个定义布局的 page master。

每个输出页面都有一个定义输出的 <fo:flow> 元素。

每个输出页面均会按序列（顺序）被打印或显示。

XSL-FO 流（Flow）

XSL-FO 页面使用来自 <fo:flow> 元素的内容进行填充。

<fo:flow> 元素包含所有被打印到页面的元素。

当页面被印满以后，相同的 page master 会被一遍又一遍地被使用，直到所有文本被打印为止。

流动到何处？

<fo:flow> 元素有一个 "flow-name" 属性。

flow-name 属性的值定义 <fo:flow> 元素的内容会去往何处。

合法的值：

- xsl-region-body（进入 region-body）
- xsl-region-before（进入 region-before）
- xsl-region-after（进入 region-after）
- xsl-region-start（进入 region-start）
- xsl-region-end（进入 region-end）

XSL-FO 页面

XSL-FO 使用名为 "Page Masters" 的页面模板来定义页面的布局。

XSL-FO 页面模板（Page Templates）

XSL-FO 使用名为 "Page Masters" 的页面模板来定义页面的布局。每个模板必须拥有一个唯一的名称：

```
<fo:simple-page-master master-name="intro">
<fo:region-body margin="5in" />
</fo:simple-page-master>

<fo:simple-page-master master-name="left">
<fo:region-body margin-left="2in" margin-right="3in" />
</fo:simple-page-master>

<fo:simple-page-master master-name="right">
<fo:region-body margin-left="3in" margin-right="2in" />
</fo:simple-page-master>
```

在上面的实例中，三个 <fo:simple-page-master> 元素，定义了三个不同的模板。每个模板（page-master）都有不同的名称。

第一个模板名为 "intro"。它可作为介绍页面的模板使用。

第二个和第三个模板名为 "left" 和 "right"。它们可作为偶数和奇数页码的页面模板使用。

XSL-FO 页面尺寸（Page Size）

XSL-FO 使用下面的属性定义页面的尺寸：

- page-width 定义页面的宽度
- page-height 定义页面的高度

XSL-FO 页面边距（Page Margins）

XSL-FO 使用下面的属性定义页面的边距：

- margin-top 定义上边距
- margin-bottom 定义下边距
- margin-left 定义左边距
- margin-right 定义右边距
- margin 定义所有边的边距

XSL-FO 页面区（Page Regions）

XSL-FO 使用下面的元素定义页面的区：

- region-body 定义主体区
- region-before 定义顶部区（页眉）
- region-after 定义底部区（页脚）
- region-start 定义左侧区（左侧栏）
- region-end 定义右侧区（右侧栏）

请注意，region-before、region-after、region-start 以及 region-end 是主体区的一部分。为了避免主体区的文本覆盖到这些区域的文本，主体区的边距至少要等于其他区的尺寸。



XSL-FO 实例

这是从某个 XSL-FO 文档中提取的一个片断：

```
<fo:simple-page-master master-name="A4" page-width="297mm"
page-height="210mm" margin-top="1cm" margin-bottom="1cm"
margin-left="1cm" margin-right="1cm">
<fo:region-body margin="3cm"/>
<fo:region-before extent="2cm"/>
<fo:region-after extent="2cm"/>
<fo:region-start extent="2cm"/>
<fo:region-end extent="2cm"/>
</fo:simple-page-master>
```

上面的代码定义了一个名称为 "A4" 的 "Simple Page Master Template"。

页面的宽度是 297 毫米，高度是 210 毫米。

页面的四个边距（上边距、下边距、左边距、右边距）均为 1 厘米。

主体的边距是 3 厘米（四个边都是）。

主体的 before、after、start 以及 end 区均为 2 厘米。

上面的实例中的主体的宽度可通过页面宽度减去左右边距以及 region-body 的边距来计算得出：

297mm - (2 x 1cm) - (2 x 3cm) = 297mm - 20mm - 60mm = 217mm

请注意，region（region-start 和 region-end）没有被计算进来。正如之前讲解过的，这些区（region）是主体的组成部分。

XSL-FO 块

XSL-FO 的输出位于块区域中。

XSL-FO 页面（Page）、流（Flow）以及块（Block）

内容"块"会"流"入"页面"中，然后输出到媒介。

XSL-FO 输出通常被嵌套在 <fo:block> 元素内，<fo:block> 嵌套于 <fo:flow> 元素内，<fo:flow> 嵌套于 <fo:page-sequence> 元素内：

```
<fo:page-sequence>
<fo:flow flow-name="xsl-region-body">
<fo:block>
<!-- Output goes here -->
</fo:block>
</fo:flow>
</fo:page-sequence>
```

块区域的属性

块是位于矩形框中的输出序列：

```
<fo:block border-width="1mm">
This block of output will have a one millimeter border around it.
</fo:block>
```

由于块区域是矩形框，所以可共享许多公共的区域属性：

- space before 和 space after
- margin
- border
- padding



space before 和 **space after** 是块与块之间起分割作用的空白。

margin 是块外侧的空白区域。

border 是区域外部边缘的矩形。其四个边均可有不同的宽度。它也可被填充为不同的颜色和背景图像。

padding 是位于 border 与 content 区域之间的区域。

content 区域可包含实际的内容，比如文本、图片、图形等等。

块边距（**Block Margin**）

- margin
- margin-top
- margin-bottom
- margin-left
- margin-right

块边框（**Block Border**）

边框样式属性：

- border-style
- border-before-style
- border-after-style
- border-start-style
- border-end-style
- border-top-style（等同于 border-before）
- border-bottom-style（等同于 border-after）
- border-left-style（等同于 border-start）
- border-right-style（等同于 border-end）

边框颜色属性：

- border-color
- border-before-color
- border-after-color
- border-start-color
- border-end-color
- border-top-color（等同于 border-before）
- border-bottom-color（等同于 border-after）
- border-left-color（等同于 border-start）
- border-right-color（等同于 border-end）

边框宽度属性：

- border-width
- border-before-width

- border-after-width
- border-start-width
- border-end-width
- border-top-width（等同于 border-before）
- border-bottom-width（等同于 border-after）
- border-left-width（等同于 border-start）
- border-right-width（等同于 border-end）

块填充（Block Padding）

- padding
- padding-before
- padding-after
- padding-start
- padding-end
- padding-top（等同于 padding-before）
- padding-bottom（等同于 padding-after）
- padding-left（等同于 padding-start）
- padding-right（等同于 padding-end）

块背景（Block Background）

- background-color
- background-image
- background-repeat
- background-attachment（scroll 或 fixed）

块样式属性（Block Styling Attributes）

块是可被单独样式化的输出序列：

```
<fo:block font-size="12pt" font-family="sans-serif">
This block of output will be written in a 12pt sans-serif font.
</fo:block>
```

字体属性：

- font-family
- font-weight
- font-style

- font-size
- font-variant

文本属性：

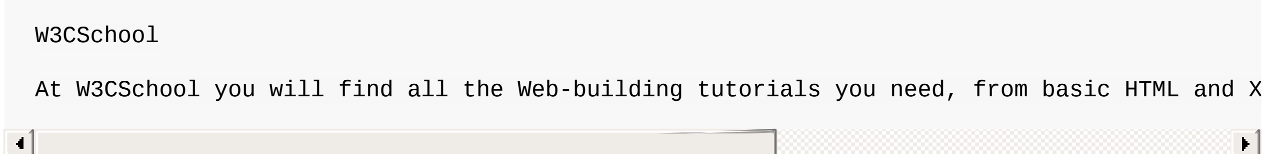
- text-align
- text-align-last
- text-indent
- start-indent
- end-indent
- wrap-option（定义自动换行）
- break-before（定义分页符）
- break-after（定义分页符）
- reference-orientation（定义 90" 增量的文字旋转）

实例

```
<fo:block font-size="14pt" font-family="verdana" color="red"
space-before="5mm" space-after="5mm">
W3CSchool
</fo:block>

<fo:block text-indent="5mm" font-family="verdana" font-size="12pt">
At W3CSchool you will find all the Web-building tutorials you
need, from basic HTML and XHTML to advanced XML, XSL, Multimedia and WAP.
</fo:block>
```

结果：



请看上面的实例，如果要生成一个拥有许多标题和段落的文档，那么将会需要非常多的代码。

通常，XSL-FO 文档不会像我们刚才所做的那样对格式化信息和内容进行组合。

通过 XSLT 的些许帮助，我们就可以把格式化信息置入模板，然后编写出更纯净的内容。

您会在本教程后面的章节学习到如何使用 XSLT 模板来组合 XSL-FO。

XSL-FO 列表

XSL-FO 使用 <fo:list-block> 元素来定义列表。

XSL-FO 列表块（List Blocks）

有四种 XSL-FO 对象可用来创建列表：

- fo:list-block（包含整个列表） (contains the whole list)
- fo:list-item（包含列表中的每个项目） (contains each item in the list)
- fo:list-item-label（包含用于 list-item 的标签 - 典型地，包含一个数字或者字符的 <fo:block> ）
- fo:list-item-body（包含 list-item 的内容/主体 - 典型地，一个或多个 <fo:block> 对象）

一个 XSL-FO 列表实例:

```
<fo:list-block>

<fo:list-item>
<fo:list-item-label>
<fo:block>*</fo:block>
</fo:list-item-label>
<fo:list-item-body>
<fo:block>Volvo</fo:block>
</fo:list-item-body>
</fo:list-item>

<fo:list-item>
<fo:list-item-label>
<fo:block>*</fo:block>
</fo:list-item-label>
<fo:list-item-body>
<fo:block>Saab</fo:block>
</fo:list-item-body>
</fo:list-item>

</fo:list-block>
```

上面代码的输出如下所示：

- Volvo
- Saab

XSL-FO 表格

XSL-FO 使用 <fo:table-and-caption> 元素来定义表格。

XSL-FO 表格（Tables）

XSL-FO 表格模型与 HTML 表格模型不是完全不同的。

有九种 XSL-FO 对象可用来创建表格：

- fo:table-and-caption
- fo:table
- fo:table-caption
- fo:table-column
- fo:table-header
- fo:table-footer
- fo:table-body
- fo:table-row
- fo:table-cell

XSL-FO 使用 **<fo:table-and-caption>** 元素来定义表格。它包含一个 <**fo:table**> 以及一个可选的 **<fo:caption>** 元素。

<fo:table> 元素包含可选的 **<fo:table-column>** 元素，一个可选的 **<fo:table-header>** 元素，一个 **<fo:table-body>** 元素，一个可选的 **<fo:table-footer>** 元素。这些元素中的每一个都可能拥有一个或多个 **<fo:table-row>** 元素，而 **<fo:table-row>** 同时会带有一个或多个 **<fo:table-cell>** 元素：

```
<fo:table-and-caption>
<fo:table>
<fo:table-column column-width="25mm"/>
<fo:table-column column-width="25mm"/>

<fo:table-header>
<fo:table-row>
<fo:table-cell>
<fo:block font-weight="bold">Car</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block font-weight="bold">Price</fo:block>
</fo:table-cell>
</fo:table-row>
</fo:table-header>

<fo:table-body>
<fo:table-row>
<fo:table-cell>
<fo:block>Volvo</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block>$50000</fo:block>
</fo:table-cell>
</fo:table-row>
<fo:table-row>
<fo:table-cell>
<fo:block>SAAB</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block>$48000</fo:block>
</fo:table-cell>
</fo:table-row>
</fo:table-body>

</fo:table>
</fo:table-and-caption>
```

以上代码的输出如下所示：

Car	Price
Volvo	$50000
SAAB	$48000

XSL-FO 与 XSLT

XSL-FO 与 XSLT 可彼此互助。

还记得这个实例吗？

```
<fo:block font-size="14pt" font-family="verdana" color="red"
space-before="5mm" space-after="5mm">
W3CSchool
</fo:block>

<fo:block text-indent="5mm" font-family="verdana" font-size="12pt">
At W3CSchool you will find all the Web-building tutorials you
need, from basic HTML and XHTML to advanced XML, XSL, Multimedia and WAP.
</fo:block>
```

结果：

```
W3CSchool

At W3CSchool you will find all the Web-building tutorials you need, from basic HTML and X
```

上面的实例来自于有关 XSL-FO 块区域的那一章节。

来自 XSLT 的帮助

从文档移除 XSL-FO 信息：

```
<header>W3CSchool</header>

<paragraph>At W3CSchool you will find all the Web-building tutorials you
need, from basic HTML and XHTML to advanced XML, XSL, Multimedia and WAP.
</paragraph>
```

添加 XSLT 转换：

```
<xsl:template match="header">
<fo:block font-size="14pt" font-family="verdana" color="red"
space-before="5mm" space-after="5mm">
<xsl:apply-templates/>
</fo:block>
</xsl:template>

<xsl:template match="paragraph">
<fo:block text-indent="5mm" font-family="verdana" font-size="12pt">
<xsl:apply-templates/>
</fo:block>
</xsl:template>
```

产生的结果是相同的：

```
W3CSchool

At W3CSchool you will find all the Web-building tutorials you need, from basic HTML and X
```

XSL-FO 软件

XSL-FO 需要格式化软件来产生输出。

XSL-FO 处理器

一个 XSL-FO 处理器是一个用于格式化输出 XSL 文档的软件程序。

大多数的 XSL-FO 处理器可以输出 HTML、PDF 文档和质量打印。

下面介绍一些知名的 XSL-FO 处理器。

Antenna House Formatter V5

Antenna House Formatter V5 是为 PDF 或打印格式化 XML 文档的软件程序。

访问 Antenna House

Altova 的 StyleVision

StyleVision 基于您的设计自动生成一致性标准的 XSLT 和 XSL:FO 样式表，以及相应的 HTML、RTF、PDF、 Word 2007 输出，等等。

访问 Altova

Ecrion 的 XF 产品

XSL-FO 格式化的一些产品！

访问 Ecrion

XSL-FO 参考手册

XSL 格式化对象参考手册

将描述转换为呈现的过程被称为格式化（formatting）。

对象	描述
basic-link	代表一个链接的起始资源。
bidi-override	重写默认 Unicode BIDI 的方向。
block	定义一个输出块（比如段落和标题）。
block-container	定义一个块级的引用区域（reference-area）。
character	规定将被映射为供呈现的字形的字符。
color-profile	定义样式表的一个颜色配置文件。
conditional-page-master-reference	规定一个当所定义的条件成立时使用的 page-master。
declarations	组合一个样式表的全局声明。
external-graphic	用于图像数据位于 XML 结果树之外的某个图形。
float	通常用于在页面起始处的一个单独区域里定位图像，或者通过将内容沿图像的一侧流动来定位图像到一侧。
flow	包含要打印到页面的所有元素。
footnote	定义在页面的 region-body 内部的一个脚注。
footnote-body	定义脚注的内容。
initial-property-set	格式化 <fo:block> 的第一行。
inline	通过背景属性或将其嵌入一个边框来定义文本的一部分格式。
inline-container	定义一个内联参考域（reference-area）。
instream-foreign-object	用于内联图形或 "generic" 类对象。在其中，对象的数据以 <fo:instream-foreign-object> 的后代形式存在。
layout-master-set	保存所有在文档中使用的宿主（master）。
leader	用于生成 "." 符号来分隔内容表格中页面数字的标题，或创建表单中的输入字段，或创建水平规则。
list-block	定义列表。

list-item	包含列表中的每个项。
list-item-body	包含了 list-item 的内容/主体。
list-item-label	包含了 list-item 标签（通常是数字、字符等）。
marker	与 <fo:retrieve-marker> 一起使用来创建运行的页眉或页脚。
multi-case	包含 XSL-FO 对象的每个供选择的子树（在 <fo:multi-switch> 内部）。父元素 <fo:multi-switch> 会选择要显示的那个选项并隐藏其余的选项。
multi-properties	用于两个或多个属性集之间切换。
multi-property-set	规定一个根据用户代理状态进行应用的可选的属性集。
multi-switch	保留一个或多个 <fo:multi-case> 对象，控制它们（由 <fo:multi-toggle> 触发）彼此之间的转换。
multi-toggle	用于切换到另一个 <fo:multi-case>。
page-number	表示当前页码。
page-number-citation	为页面引用页码，此页面包含由被引用对象返回的第一个正常区域。
page-sequence	页面输出元素的容器。每个页面布局将有一个 <fo:page-sequence> 对象。
page-sequence-master	规定要使用的 simple-page-masters 以及使用顺序。
region-after	定义页脚。
region-before	定义页眉。
region-body	定义页面主题。
region-end	定义页面的右侧栏。
region-start	定义页面的左侧栏。
repeatable-page-master-alternatives	规定一组 simple-page-master 的副本。
repeatable-page-master-reference	规定单个 simple-page-master 的副本。
retrieve-marker	与 <fo:marker> 一起使用来创建运行的页眉或页脚。
root	XSL-FO 文档的根（顶级）节点。
simple-page-master	定义一个页面的尺寸和形状。
single-page-	

master-reference	规定用在页面序列的给定点中的 page-master。
static-content	对象包含了静态内容（如：页眉和页脚），该静态内容将在多个页面中重复调用。
table	格式化表格的表格式材料。
table-and-caption	格式化表格及其标题。
table-body	包含表格行和表格单元格的容器。
table-caption	包含表格的标题。
table-cell	定义表格单元格。
table-column	格式化表格的列。
table-footer	定义表格的页脚。
table-header	定义表格的页眉。
table-row	定义表格行。
title	为一个 page-sequence 定义一个标题。
wrapper	为一组 XSL-FO 对象规定 inherited[继承] 属性。

SVG 教程

SVG 简介

SVG 是使用 XML 来描述二维图形和绘图程序的语言。

学习之前应具备的基础知识：

继续学习之前，你应该对以下内容有基本的了解：

- HTML
- XML 基础

如果希望首先学习这些内容，请在本站的首页选择相应的教程。

什么是SVG？

- SVG 指可伸缩矢量图形 (Scalable Vector Graphics)
- SVG 用来定义用于网络的基于矢量的图形
- SVG 使用 XML 格式定义图形
- SVG 图像在放大或改变尺寸的情况下其图形质量不会有所损失
- SVG 是万维网联盟的标准
- SVG 与诸如 DOM 和 XSL 之类的 W3C 标准是一个整体

SVG 是 W3C 推荐标准

SVG 于 2003 年 1 月 14 日成为 W3C 推荐标准。

如需阅读更多有关 W3C 的 SVG 活动的信息，请访问我们的 W3C 教程。

SVG 的历史和优势

在 2003 年一月，SVG 1.1 被确立为 W3C 标准。

参与定义 SVG 的组织有：太阳微系统、Adobe、苹果公司、IBM 以及柯达。

与其他图像格式相比，使用 SVG 的优势在于：

- SVG 可被非常多的工具读取和修改（比如记事本）
- SVG 与 JPEG 和 GIF 图像比起来，尺寸更小，且可压缩性更强。
- SVG 是可伸缩的
- SVG 图像可在任何的分辨率下被高质量地打印

- SVG 可在图像质量不下降的情况下被放大
- SVG 图像中的文本是可选的，同时也是可搜索的（很适合制作地图）
- SVG 可以与 Java 技术一起运行
- SVG 是开放的标准
- SVG 文件是纯粹的 XML

SVG 的主要竞争者是 Flash。

与 Flash 相比，SVG 最大的优势是与其他标准（比如 XSL 和 DOM）相兼容。而 Flash 则是未开源的私有技术。

查看 SVG 文件

Internet Explorer9，火狐，谷歌Chrome，Opera和Safari都支持SVG。

IE8和早期版本都需要一个插件 - 如Adobe SVG浏览器，这是免费提供的。

创建SVG文件

由于SVG是XML文件，SVG图像可以用任何文本编辑器创建，但它往往是与一个绘图程序一起使用，如Inkscape，更方便地创建SVG图像。

SVG 实例

简单的 SVG 实例

一个简单的SVG图形例子：

这里是SVG文件（SVG文件的保存与SVG扩展）：

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <circle cx="100" cy="50" r="40" stroke="black"
  stroke-width="2" fill="red" />
</svg>
```

SVG 代码解析：

第一行包含了 XML 声明。请注意 standalone 属性！该属性规定此 SVG 文件是否是"独立的"，或含有对外部文件的引用。

standalone="no" 意味着 SVG 文档会引用一个外部文件 - 在这里，是 DTD 文件。

第二和第三行引用了这个外部的 SVG DTD。该 DTD 位于 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd"。该 DTD 位于 W3C，含有所有允许的 SVG 元素。

SVG 代码以 <svg> 元素开始，包括开启标签 <svg> 和关闭标签 </svg> 。这是根元素。width 和 height 属性可设置此 SVG 文档的宽度和高度。version 属性可定义所使用的 SVG 版本，xmlns 属性可定义 SVG 命名空间。

SVG 的 <circle> 用来创建一个圆。cx 和 cy 属性定义圆中心的 x 和 y 坐标。如果忽略这两个属性，那么圆点会被设置为 (0, 0)。r 属性定义圆的半径。

stroke 和 stroke-width 属性控制如何显示形状的轮廓。我们把圆的轮廓设置为 2px 宽，黑边框。

fill 属性设置形状内的颜色。我们把填充颜色设置为红色。

关闭标签的作用是关闭 SVG 元素和文档本身。

注释：所有的开启标签必须有关闭标签！

SVG 在 HTML 页面

SVG 文件可通过以下标签嵌入 HTML 文档：<embed>、<object> 或者 <iframe>。

SVG的代码可以直接嵌入到HTML页面中，或您可以直接链接到SVG文件。

使用 <embed> 标签

<embed>:

- 优势：所有主要浏览器都支持，并允许使用脚本
- 缺点：不推荐在HTML4和XHTML中使用（但在HTML5允许）

语法**:**

```
<embed src="circle1.svg" type="image/svg+xml" />
```

结果：

使用 <object> 标签

<object>:

- 优势：所有主要浏览器都支持，并支持HTML4，XHTML和HTML5标准
- 缺点：不允许使用脚本。

语法**:**

```
<object data="circle1.svg" type="image/svg+xml"></object>
```

结果：

使用 <iframe> 标签

<iframe>:

- 优势：所有主要浏览器都支持，并允许使用脚本
- 缺点：不推荐在HTML4和XHTML中使用（但在HTML5允许）

语法**:**

```
<iframe src="circle1.svg"></iframe>
```

结果:

直接在HTML嵌入SVG代码

在Firefox、Internet Explorer9、谷歌Chrome和Safari中，你可以直接在HTML嵌入SVG代码。

注意：SVG不能直接嵌入到Opera。

实例

```
<html>
<body>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <circle cx="100" cy="50" r="40" stroke="black"
  stroke-width="2" fill="red"/>
</svg>

</body>
</html>
```

链接到SVG文件

您还可以用<a>标签链接到一个SVG文件：链接到SVG文件

您还可以用<a>标签链接到一个SVG文件：

```
<a href="circle1.svg">View SVG file</a>
```

结果:

查看 SVG 文件

SVG <rect>

SVG Shapes

SVG有一些预定义的形状元素，可被开发者使用和操作：

- 矩形 <rect>
- 圆形 <circle>
- 椭圆 <ellipse>
- 线 <line>
- 折线 <polyline>
- 多边形 <polygon>
- 路径 <path>

下面的章节会为您讲解这些元素，首先从矩形元素开始。

SVG 矩形 - <rect>

实例 1

<rect> 标签可用来创建矩形，以及矩形的变种：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect width="300" height="100"
  style="fill:rgb(0,0,255);stroke-width:1;stroke:rgb(0,0,0)"/>
</svg>
```

对于Opera用户： 查看SVG文件（右键单击SVG图形预览源）。

代码解析:

- rect 元素的 width 和 height 属性可定义矩形的高度和宽度
- style 属性用来定义 CSS 属性
- CSS 的 fill 属性定义矩形的填充颜色（rgb 值、颜色名或者十六进制值）
- CSS 的 stroke-width 属性定义矩形边框的宽度
- CSS 的 stroke 属性定义矩形边框的颜色

实例 2

让我们看看另一个例子，它包含一些新的属性：

Here is the SVG code:

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect x="50" y="20" width="150" height="150"
  style="fill:blue;stroke:pink;stroke-width:5;fill-opacity:0.1;
  stroke-opacity:0.9"/>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- x 属性定义矩形的左侧位置（例如，x="0" 定义矩形到浏览器窗口左侧的距离是 0px）
- y 属性定义矩形的顶端位置（例如，y="0" 定义矩形到浏览器窗口顶端的距离是 0px）
- CSS 的 fill-opacity 属性定义填充颜色透明度（合法的范围是：0 - 1）
- CSS 的 stroke-opacity 属性定义笔触颜色的透明度（合法的范围是：0 - 1）

实例 3

定义整个元素的不透明度：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect x="50" y="20" width="150" height="150"
  style="fill:blue;stroke:pink;stroke-width:5;opacity:0.5"/>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

- The CSS opacity property defines the opacity value for the whole element (legal range: 0 to 1)

实例 4

最后一个例子，创建一个圆角矩形：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect x="50" y="20" rx="20" ry="20" width="150" height="150"
  style="fill:red;stroke:black;stroke-width:5;opacity:0.5"/>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

- rx 和 ry 属性可使矩形产生圆角。

SVG <circle>

SVG 圆形 - <circle>

<circle> 标签可用来创建一个圆：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <circle cx="100" cy="50" r="40" stroke="black"
  stroke-width="2" fill="red"/>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- cx和cy属性定义圆点的x和y坐标。如果省略cx和cy，圆的中心会被设置为(0, 0)
- r属性定义圆的半径

SVG <ellipse>

SVG 椭圆 - <ellipse>

实例 1

lt;ellipse> 元素是用来创建一个椭圆：

椭圆与圆很相似。不同之处在于椭圆有不同的x和y半径，而圆的x和y半径是相同的：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <ellipse cx="300" cy="80" rx="100" ry="50"
  style="fill:yellow;stroke:purple;stroke-width:2"/>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- CX属性定义的椭圆中心的x坐标
- CY属性定义的椭圆中心的y坐标
- RX属性定义的水平半径
- RY属性定义的垂直半径

实例 2

下面的例子创建了三个累叠而上的椭圆：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <ellipse cx="240" cy="100" rx="220" ry="30" style="fill:purple"/>
  <ellipse cx="220" cy="70" rx="190" ry="20" style="fill:lime"/>
  <ellipse cx="210" cy="45" rx="170" ry="15" style="fill:yellow"/>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

实例 3

下面的例子组合了两个椭圆（一个黄的和一个白的）：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <ellipse cx="240" cy="50" rx="220" ry="30" style="fill:yellow"/>
  <ellipse cx="220" cy="50" rx="190" ry="20" style="fill:white"/>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

SVG <line>

SVG 直线 - <line>

<line> 元素是用来创建一个直线：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <line x1="0" y1="0" x2="200" y2="200"
  style="stroke:rgb(255,0,0);stroke-width:2"/>
</svg>
```

对于Opera用户：查看SVG文件右键单击SVG图形预览源）。

- x1 属性在 x 轴定义线条的开始
- y1 属性在 y 轴定义线条的开始
- x2 属性在 x 轴定义线条的结束
- y2 属性在 y 轴定义线条的结束

SVG <polygon>

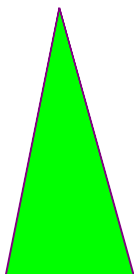
SVG 多边形 - <polygon>

实例 1

<polygon> 标签用来创建含有不少于三个边的图形。

多边形是由直线组成，其形状是"封闭"的（所有的线条 连接起来）。

 polygon来自希腊。 "Poly" 一位 "many" ， "gon" 意味 "angle".



下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <polygon points="200,10 250,190 160,210"
  style="fill:lime;stroke:purple;stroke-width:1"/>
</svg>
```

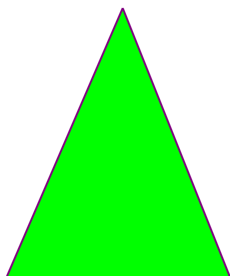
对于Opera用户： 查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- points 属性定义多边形每个角的 x 和 y 坐标

实例 2

下面的示例创建一个四边的多边形：



下面是SVG代码：

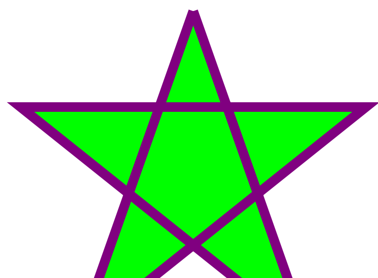
实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <polygon points="220,10 300,210 170,250 123,234"
  style="fill:lime;stroke:purple;stroke-width:1"/>
</svg>
```

对于Opera用户： 查看SVG文件（右键单击SVG图形预览源）。

实例 3

使用 <polygon> 元素创建一个星型:



下面是SVG代码：

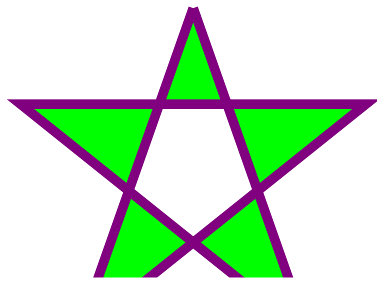
实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <polygon points="100,10 40,180 190,60 10,60 160,180"
  style="fill:lime;stroke:purple;stroke-width:5;fill-rule:nonzero;" />
</svg>
```

对于Opera用户： 查看SVG文件（右键单击SVG图形预览源）。

实例 4

改变 fill-rule 属性为 "evenodd":



下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <polygon points="100,10 40,180 190,60 10,60 160,180"
  style="fill:lime;stroke:purple;stroke-width:5;fill-rule:evenodd;" />
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

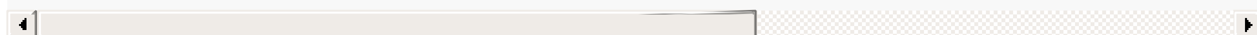
SVG <polyline>

SVG 曲线 - <polyline>

实例 1

<polyline> 元素是用于创建任何只有直线的形状：

```
<object height="200" width="500" data="polyline1.svg" type="image/svg+xml"><embed src="im
```



下面是SVG代码：

实例

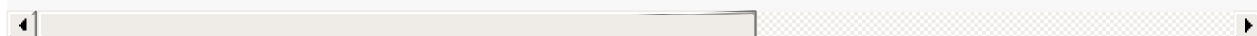
```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <polyline points="20,20 40,25 60,40 80,120 120,140 200,180"
  style="fill:none;stroke:black;stroke-width:3" />
</svg>
```

F对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

实例 2

只有直线的另一个例子：

```
<object height="200" width="500" data="polyline1.svg" type="image/svg+xml"><embed src="im
```



下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <polyline points="0,40 40,40 40,80 80,80 80,120 120,120 120,160"
  style="fill:white;stroke:red;stroke-width:4"/>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

SVG <path>

SVG 路径 - <path>

<path> 元素用于定义一个路径。

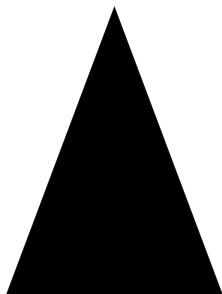
下面的命令可用于路径数据：

- M = moveto
- L = lineto
- H = horizontal lineto
- V = vertical lineto
- C = curveto
- S = smooth curveto
- Q = quadratic Bézier curve
- T = smooth quadratic Bézier curveto
- A = elliptical Arc
- Z = closepath

注意：以上所有命令均允许小写字母。大写表示绝对定位，小写表示相对定位。

实例 1

上面的例子定义了一条路径，它开始于位置150 0，到达位置75 200，然后从那里开始到225 200，最后在150 0关闭路径。



下面是SVG代码：

实例

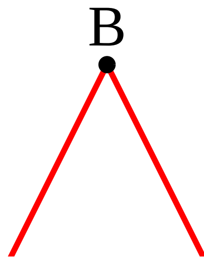
```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <path d="M150 0 L75 200 L225 200 Z" />
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

实例 2

使用贝兹曲线流畅的曲线模型，可无限期的缩放。一般情况下，用户选择两个端点和一个或两个控制点。一个一个控制点的贝塞尔曲线被称为二次贝塞尔曲线和两个控制点的那种被称为立方体。

下面的例子创建了一个二次贝塞尔曲线，A和C分别是起点和终点，B是控制点：



下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <path id="lineAB" d="M 100 350 l 150 -300" stroke="red"
  stroke-width="3" fill="none" />
  <path id="lineBC" d="M 250 50 l 150 300" stroke="red"
  stroke-width="3" fill="none" />
  <path d="M 175 200 l 150 0" stroke="green" stroke-width="3"
  fill="none" />
  <path d="M 100 350 q 150 -300 300 0" stroke="blue"
  stroke-width="5" fill="none" />
  <!-- Mark relevant points -->
  <g stroke="black" stroke-width="3" fill="black">
    <circle id="pointA" cx="100" cy="350" r="3" />
    <circle id="pointB" cx="250" cy="50" r="3" />
    <circle id="pointC" cx="400" cy="350" r="3" />
  </g>
  <!-- Label the points -->
  <g font-size="30" font="sans-serif" fill="black" stroke="none"
  text-anchor="middle">
    <text x="100" y="350" dx="-30">A</text>
    <text x="250" y="50" dy="-10">B</text>
    <text x="400" y="350" dx="30">C</text>
  </g>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

复杂吗？是的！！由于在绘制路径时的复杂性，强烈建议使用SVG编辑器来创建复杂的图形。

SVG <text>

SVG 文本 - <text>

<text> 元素用于定义文本。

实例 1

写一个文本：

I love SVG

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <text x="0" y="15" fill="red">I love SVG</text>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

实例 2

旋转的文字：

I love SVG

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <text x="0" y="15" fill="red" transform="rotate(30 20,40)">I love SVG</text>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

实例 3

路径上的文字：

I love SVG

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
xmlns:xlink="http://www.w3.org/1999/xlink">
   <defs>
    <path id="path1" d="M75,20 a1,1 0 0,0 100,0" />
  </defs>
  <text x="10" y="100" style="fill:red;">
    <textPath xlink:href="#path1">I love SVG I love SVG</textPath>
  </text>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

实例 4

元素可以安排任何分小组与<tspan> 元素的数量。每个<tspan> 元素可以包含不同的格式和位置。几行文本(与 <tspan> 元素):

I love SVG

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <text x="10" y="20" style="fill:red;">Several lines:
    <tspan x="10" y="45">First line</tspan>
    <tspan x="10" y="70">Second line</tspan>
  </text>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

实例 5

作为链接文本（ <a> 元素）：

I love SVG

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <a xlink:href="http://www.w3schools.com/svg/" target="_blank">
    <text x="0" y="15" fill="red">I love SVG</text>
  </a>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

SVG Stroke 属性

SVG Stroke 属性

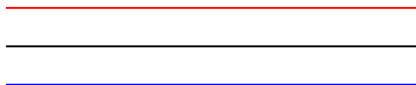
SVG提供了一个范围广泛stroke 属性。在本章中，我们将看看下面：

- stroke
- stroke-width
- stroke-linecap
- stroke-dasharray

所有stroke属性，可应用于任何种类的线条，文字和元素就像一个圆的轮廓。

SVG stroke 属性

Stroke属性定义一条线，文本或元素轮廓颜色：



下面是SVG代码：

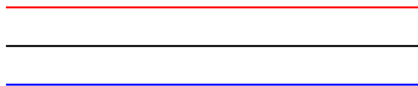
实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <g fill="none">
    <path stroke="red" d="M5 20 l215 0" />
    <path stroke="blue" d="M5 40 l215 0" />
    <path stroke="black" d="M5 60 l215 0" />
  </g>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

SVG stroke-width 属性

Tstroke- width属性定义了一条线，文本或元素轮廓厚度：



下面是SVG代码：

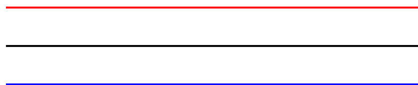
实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <g fill="none" stroke="black">
    <path stroke-width="2" d="M5 20 l215 0" />
    <path stroke-width="4" d="M5 40 l215 0" />
    <path stroke-width="6" d="M5 60 l215 0" />
  </g>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

SVG stroke-linecap 属性

strokelinecap属性定义不同类型的开放路径的终结：



下面是SVG代码：

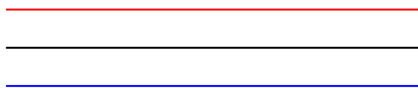
实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <g fill="none" stroke="black" stroke-width="6">
    <path stroke-linecap="butt" d="M5 20 l215 0" />
    <path stroke-linecap="round" d="M5 40 l215 0" />
    <path stroke-linecap="square" d="M5 60 l215 0" />
  </g>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

SVG stroke-dasharray 属性

strokedasharray属性用于创建虚线：



下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <g fill="none" stroke="black" stroke-width="4">
    <path stroke-dasharray="5,5" d="M5 20 l215 0" />
    <path stroke-dasharray="10,10" d="M5 40 l215 0" />
    <path stroke-dasharray="20,10,5,5,5,10" d="M5 60 l215 0" />
  </g>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

SVG 滤镜

SVG滤镜用来增加对SVG图形的特殊效果。

SVG 滤镜

在本教程中，我们将仅展示一个可能采用的特殊效果。基础知识展示后，你已经学会使用特殊效果，你应该能够适用于其他地方。这里的关键是给你一个怎样做SVG的想法，而不是重复整个规范。

SVG可用的滤镜是：

- feBlend - 与图像相结合的滤镜
- feColorMatrix - 用于彩色滤光片转换
- feComponentTransfer
- feComposite
- feConvolveMatrix
- feDiffuseLighting
- feDisplacementMap
- feFlood
- feGaussianBlur
- feImage
- feMerge
- feMorphology
- feOffset - 过滤阴影
- feSpecularLighting
- feTile
- feTurbulence
- feDistantLight - 用于照明过滤
- fePointLight - 用于照明过滤
- feSpotLight - 用于照明过滤



> 除此之外，您可以在每个 SVG 元素上使用多个滤镜！

SVG 模糊效果

注意： Internet Explorer和Safari不支持SVG滤镜！

<defs> 和 <filter>

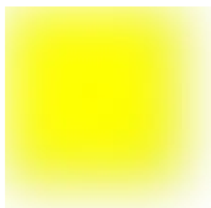
所有互联网的SVG滤镜定义在<defs>元素中。<defs>元素定义短并含有特殊元素（如滤镜）定义。

<filter>标签用来定义SVG滤镜。<filter>标签使用必需的id属性来定义向图形应用哪个滤镜？

SVG <feGaussianBlur>

实例 1

<feGaussianBlur> 元素是用于创建模糊效果：



下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <filter id="f1" x="0" y="0">
      <feGaussianBlur in="SourceGraphic" stdDeviation="15" />
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green" stroke-width="3"
  fill="yellow" filter="url(#f1)" />
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- <filter>元素id属性定义一个滤镜的唯一名称
- <feGaussianBlur>元素定义模糊效果

- in="SourceGraphic"这个部分定义了由整个图像创建效果
- stdDeviation属性定义模糊量
- <rect>元素的滤镜属性用来把元素链接到"f1"滤镜

SVG 阴影

注意： Internet Explorer和Safari不支持SVG滤镜！

<defs> 和 <filter>

所有互联网的SVG滤镜定义在<defs>元素中。<defs>元素定义短并含有特殊元素（如滤镜）定义。

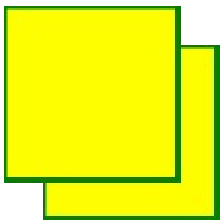
<filter>标签用来定义SVG滤镜。<filter>标签使用必需的id属性来定义向图形应用哪个滤镜？

SVG <feOffset>

实例 1

<feOffset>元素是用于创建阴影效果。我们的想法是采取一个SVG图形（图像或元素）并移动它在xy平面上一点儿。

第一个例子偏移一个矩形（带<feOffset>），然后混合偏移图像顶部（含<feBlend>）：



下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <filter id="f1" x="0" y="0" width="200%" height="200%">
      <feOffset result="offOut" in="SourceGraphic" dx="20" dy="20" />
      <feBlend in="SourceGraphic" in2="offOut" mode="normal" />
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green" stroke-width="3"
  fill="yellow" filter="url(#f1)" />
</svg>
```

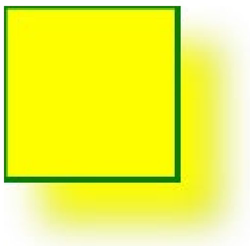
对于Opera用户： 查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- <filter>元素id属性定义一个滤镜的唯一名称
- <rect>元素的滤镜属性用来把元素链接到"f1"滤镜

实例 2

现在，偏移图像可以变的模糊（含 <feGaussianBlur>）：



下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <filter id="f1" x="0" y="0" width="200%" height="200%">
      <feOffset result="offOut" in="SourceGraphic" dx="20" dy="20" />
      <feGaussianBlur result="blurOut" in="offOut" stdDeviation="10" />
      <feBlend in="SourceGraphic" in2="blurOut" mode="normal" />
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green" stroke-width="3"
  fill="yellow" filter="url(#f1)" />
</svg>
```

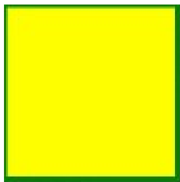
对于Opera用户： 查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- <feGaussianBlur>元素的stdDeviation属性定义了模糊量

实例 3

现在，制作一个黑色的阴影：



下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <filter id="f1" x="0" y="0" width="200%" height="200%">
      <feOffset result="offOut" in="SourceAlpha" dx="20" dy="20" />
      <feGaussianBlur result="blurOut" in="offOut" stdDeviation="10" />
      <feBlend in="SourceGraphic" in2="blurOut" mode="normal" />
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green" stroke-width="3"
  fill="yellow" filter="url(#f1)" />
</svg>
```

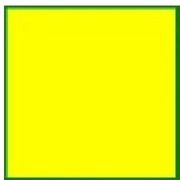
对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- <feOffset>元素的属性改为"SourceAlpha"在Alpha通道使用残影，而不是整个RGBA像素。

实例 4

现在为阴影涂上一层颜色：



下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <filter id="f1" x="0" y="0" width="200%" height="200%">
      <feOffset result="offOut" in="SourceGraphic" dx="20" dy="20" />
      <feColorMatrix result="matrixOut" in="offOut" type="matrix"
      values="0.2 0 0 0 0 0 0.2 0 0 0 0 0 0.2 0 0 0 0 0 1 0" />
      <feGaussianBlur result="blurOut" in="matrixOut" stdDeviation="10" />
      <feBlend in="SourceGraphic" in2="blurOut" mode="normal" />
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green" stroke-width="3"
  fill="yellow" filter="url(#f1)" />
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- <feColorMatrix>过滤器是用来转换偏移的图像使之更接近黑色的颜色。'0.2'矩阵的三个值都获取乘以红色，绿色和蓝色通道。降低其值带来的颜色至黑色（黑色为0）

SVG 渐变 - 线性

SVG 渐变

渐变是一种从一种颜色到另一种颜色的平滑过渡。另外，可以把多个颜色的过渡应用到同一个元素上。

SVG渐变主要有两种类型：

- Linear
- Radial

SVG 线性渐变 - <linearGradient>

<linearGradient>元素用于定义线性渐变。

<linearGradient>标签必须嵌套在<defs>的内部。<defs>标签是definitions的缩写，它可对诸如渐变之类的特殊元素进行定义。

线性渐变可以定义为水平，垂直或角渐变：

- 当y1和y2相等，而x1和x2不同时，可创建水平渐变
- 当x1和x2相等，而y1和y2不同时，可创建垂直渐变
- 当x1和x2不同，且y1和y2不同时，可创建角形渐变

实例 1

定义水平线性渐变从黄色到红色的椭圆形：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <linearGradient id="grad1" x1="0%" y1="0%" x2="100%" y2="0%">
      <stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
      <stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
    </linearGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad1)" />
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- <linearGradient>标签的id属性可为渐变定义一个唯一的名称
- <linearGradient>标签的X1，X2，Y1，Y2属性定义渐变开始和结束位置
- 渐变的颜色范围可由两种或多种颜色组成。每种颜色通过一个<stop>标签来规定。offset属性用来定义渐变的开始和结束位置。
- 填充属性把 ellipse 元素链接到此渐变

实例 2

定义一个垂直线性渐变从黄色到红色的椭圆形：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <linearGradient id="grad1" x1="0%" y1="0%" x2="0%" y2="100%">
      <stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
      <stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
    </linearGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad1)" />

</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

实例 3

定义一个椭圆形，水平线性渐变从黄色到红色并添加一个椭圆内文本：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <linearGradient id="grad1" x1="0%" y1="0%" x2="100%" y2="0%">
      <stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
      <stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
    </linearGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad1)" />
  <text fill="#ffffff" font-size="45" font-family="Verdana" x="150" y="86">
  SVG</text>
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- <text> 元素是用来添加一个文本

SVG 渐变 - 放射性

SVG 放射性渐变 - <radialGradient>

<radialGradient>元素用于定义放射性渐变。

<radialGradient>标签必须嵌套在<defs>的内部。<defs>标签是definitions的缩写，它可对诸如渐变之类的特殊元素进行定义。

实例 1

定义一个放射性渐变从白色到蓝色椭圆：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <radialGradient id="grad1" cx="50%" cy="50%" r="50%" fx="50%" fy="50%">
      <stop offset="0%" style="stop-color:rgb(255,255,255);
      stop-opacity:0" />
      <stop offset="100%" style="stop-color:rgb(0,0,255);stop-opacity:1" />
    </radialGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad1)" />
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

代码解析：

- <radialGradient>标签的 id 属性可为渐变定义一个唯一的名称
- CX，CY和r属性定义的最外层圆和Fx和Fy定义的最内层圆
- 渐变颜色范围可以由两个或两个以上的颜色组成。每种颜色用一个<stop>标签指定。offset属性用来定义渐变色开始和结束
- 填充属性把ellipse元素链接到此渐变

实例 2

定义放射性渐变从白色到蓝色的另一个椭圆：

下面是SVG代码：

实例

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <radialGradient id="grad1" cx="20%" cy="30%" r="30%" fx="50%" fy="50%">
      <stop offset="0%" style="stop-color:rgb(255,255,255);
      stop-opacity:0" />
      <stop offset="100%" style="stop-color:rgb(0,0,255);stop-opacity:1" />
    </radialGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad1)" />
</svg>
```

对于Opera用户：查看SVG文件（右键单击SVG图形预览源）。

SVG 参考手册

SVG 元素

元素	说明	属性
<a>	创建一个SVG元素周围链接	xlink:show xlink:actuate xlink:href targe
<altGlyph>	允许对象性文字进行控制，来呈现特殊的字符数据	x y dx dy rotate glyphRef format xlink:hr
<altGlyphDef>	定义一系列象性符号的替换	id
<altGlyphItem>	定义一系列候选的象性符号的替换	id
<animate>	随时间动态改变属性	attributeName="目标属性名称" from="起 dur="持续时间" repeatCount="动画时间
<animateColor>	定义随着时间的推移颜色转换	by="相对偏移值" from="起始值" to="结束
<animateMotion>	使元素沿着动作路径移动	calcMode="动画的插补模式。可以是'dis 'paced', 'spline'" path="运动路径" keyPo 的对象目前时间应移动多远" xlink:href=" <path>元素，它定义运动路径"
<animateTransform>	动画上一个目标元素变换属性，从而使动画控制平移，缩放，旋转或倾斜	by="相对偏移值" from="起始值" to="结束 转换其值是随时间变化。可以是 'transla 'skewX', 'skewY'"
<circle>	定义一个圆	cx="圆的x轴坐标" cy="圆的y轴坐标" r=" 现属性：颜色，FillStroke，图形
<clipPath>	用于隐藏位于剪切路径以外的对象部分。定义绘制什么和什么不绘制的模具被称为剪切路径	clip-path="引用剪贴路径和引用剪贴路径 clipPathUnits="userSpaceOnUse'或'obj 第二个值childern一个对象的边框，会使 单位（默认："userSpaceOnUse"）"
<color-profile>	指定颜色配置文件的说明（使用CSS样式文件时）	local="本地存储颜色配置文件唯一ID" na intent="auto|perceptual|relative-colorimetric|saturation|absolute-colorim xlink:href="ICC配置文件资源URI"
<cursor>	定义一个独立于平台的自定义光标	x="x轴左上角光标（默认为0）" y="y轴的 认为0）" xlink:href="使用光标图像URI

<defs>	引用的元素容器	
<desc>	对 SVG 中的元素的纯文本描述 - 并不作为图形的一部分来显示。用户代理会将其显示为工具提示	
<ellipse>	定义一个椭圆	cx="椭圆x轴坐标" cy="椭圆y轴坐标" rx= 径"。必需。 ry="沿y轴长椭圆形的半径" 性：颜色，FillStroke，图形
<feBlend>	使用不同的混合模式把两个对象合成在一起	mode="图像混合模式： normal|multiply|screen|darken|lighten" i 镜原始输入：SourceGraphic | SourceA BackgroundImage | BackgroundAlpha | StrokePaint | <filter-primitive-reference> 像的混合操作"
feColorMatrix	SVG滤镜。适用矩阵转换	
feComponentTransfer	SVG 滤镜。执行数据的 component-wise 重映射	
feComposite	SVG 滤镜	
feConvolveMatrix	SVG 滤镜	
feDiffuseLighting	SVG 滤镜	
feDisplacementMap	SVG 滤镜	
feDistantLight	SVG滤镜。定义一个光源	
feFlood	SVG滤镜	
feFuncA	SVG 滤镜。feComponentTransfer 的子元素	
feFuncB	SVG 滤镜。feComponentTransfer 的子元素	
feFuncG	SVG 滤镜。feComponentTransfer 的子元素	
feFuncR	SVG 滤镜。feComponentTransfer 的子元素	
feGaussianBlur	SVG滤镜。执行高斯模糊图像	

feImage	SVG滤镜。	
feMerge	SVG滤镜。建立在彼此顶部图像层	
feMergeNode	SVG 滤镜。feMerge的子元素	
feMorphology	SVG 滤镜。 对源图形执行"fattening" 或者"thinning"	
feOffset	SVG滤镜。相对其当前位置移动图像	
fePointLight	SVG滤镜	
feSpecularLighting	SVG滤镜	
feSpotLight	SVG滤镜	
feTile	SVG滤镜	
feTurbulence	SVG滤镜	
filter	滤镜效果的容器	
font	定义字体	
font-face	描述一种字体的特点	
font-face-format		
font-face-name		
font-face-src		
font-face-uri		
foreignObject		
<g>	用于把相关元素进行组合的容器元素	id="该组的名称" fill="该组填充颜色" opa 度" 显现属性: All
glyph	为给定的象形符号定义图形	
glyphRef	定义要使用的可能的象形符号	
hkern		
<image>	定义图像	x="图像的左上角的x轴坐标" y="图像的 width="图像的宽度". 必须. height="图像 xlink:href="图像的路径". 必须. 显现属性 Images, Viewports
<line>	定义一条线	x1="直线起始点x坐标" y1="直线起始点 点x坐标" y2="直线终点y坐标" 显现属性 Graphics, Markers

<linearGradient>	定义线性渐变。通过使用矢量线性渐变填充对象，并可以定义为水平，垂直或角渐变。	id="id 属性可为渐变定义一个唯一的名称 gradientUnits="'userSpaceOnUse' or 'o 使用视图框或对象，以确定相对位置矢量 为'objectBoundingBox）" gradientTrans 的转变" x1="渐变向量x启动点（默认0% 启动点（默认0%）" x2="渐变向量x的终 100%）" y2="渐变向量y的终点。（默 spreadMethod="'pad' or 'reflect' or 'repe xlink:href="reference to another gradien values are used as defaults and stops i Recursive"
<marker>	标记可以放在直线，折线，多边形和路径的顶点。这些元素可以使用maeker属性的"maeker-start"，"maeker-mid"和"maeker-end"，继承默认情况下或可设置为"none"或定义的标记的URI。您必须先定义标记，然后才可以通过其URI引用。任何一种形状，可以把标记放在里面。他们绘制元素时把它们附加到顶部	markerUnits="strokeWidth'或'userSpac strokeWidth"那么使用一个单位等于一个 标记尺度不会使用同一视图单位作为引 为'strokeWidth'）" refx="标记顶点连接的 refy="标记顶点连接的位置（默认为0）' 显示标记的角度。"auto"将计算某个角度 的正切值（默认为0） markerWidth="标 3）" markerHeight="标记的高度（默认 点"看到"这个SVG绘图区域。由空格或逗 (min x, min y, width, height)" presentatio
<mask>	度屏蔽是一种不透明度值的组合和裁剪。像裁剪，您可以使用图形，文字或路径定义掩码的部分。一个掩码的默认状态是完全透明的，也就是裁剪平面的对面的。在掩码的图形设置掩码的不透明部分	maskUnits="'userSpaceOnUse' or 'obje 定裁剪面是否是相对完整的视窗或对象 认：'objectBoundingBox'）" maskConte 码相对对象的图形位置使用百分 比'userSpaceOnUse'或'objectBounding 认：'userSpaceOnUse'）" x="裁剪面掩 值：-10%）" y="裁剪面掩码（默认值： 剪面掩码（默认是：120%）" height="裁 是：120%）"
metadata	指定元数据	
missing-glyph		
mpath		
<path>	定义一个路径	d="定义路径指令" pathLength="如果存在 放，以便计算各点相当于此值的路径长度 列表" 显现属性: Color, FillStroke, Graph
		id="用于引用这个模式的唯一ID。"必需的 patternUnits="userSpaceOnUse'或'obje 第二个值X，Y，width，height 一个会使 的小部分，单位（%）。"

<pattern>	视图显示和视图的大小。然后添加到您的模式形状。该模式命中时重复视图框的边缘（可视范围）	patternContentUnits="'userSpaceOnUs 'objectBoundingBox'" patternTransform 进行转换" x="模式的偏移量，来自左上 y="模式的偏移量，来自左上角（默认为 铺的宽度（默认为100%）" height="模 为100%）" viewBox="各点"看到"这个S 格或逗号分隔的4个值。(min x, min y, w xlink:href="另一种模式，其属性值是默 以继承。递归"
<polygon>	定义一个包含至少三边图形	points="多边形的点。点的总数必须是偶 rule="FillStroke演示属性的部分" 显现属 FillStroke, Graphics, Markers
<polyline>	定义只有直线组成的任意形状	points=折线上的"点"。必需的。 显现属 FillStroke, Graphics, Markers
<radialGradient>	定义放射性渐变。放射性渐变创建一个圆圈	gradientUnits="'userSpaceOnUse' or 'o 使用视图框或对象以确定相对位置的矢 为'objectBoundingBox）" gradientTrans 的变换" cx="渐变的中心点（数字或% - cy="渐变的中心点。（默认50%）" r=" 认50%）" fx="渐变的焦点。（默认0% 点。（默认0%）" spreadMethod="'pa 'repeat'" xlink:href="引用到另一个渐变， 值。递归"
<rect>	定义一个矩形	x="矩形的左上角的x轴" y="矩形的左上 半径（round元素）" ry="y轴的半径（ro width="矩形的宽度"。必需的。 height=' 的。 显现属性: Color, FillStroke, Graphi
script	脚本容器。（例如ECMAScript）	
set	设置一个属性值指定时间	
<stop>	渐变停止	offset="偏移停止（0到1/0%到100%）" color="这个stop的颜色" stop-opacity=" (0到1)"
style	可使样式表直接嵌入SVG内容内部	
<svg>	创建一个SVG文档片段	x="左上角嵌入（默认为0）" y="左上角 width="SVG片段的宽度（默认为100%） 段的高度（默认为100%）" viewBox=" 图区域。由空格或逗号分隔的4个值。（ height)" preserveAspectRatio="'none'或 9种组合,VAL是"min"，"mid"或"max"。 none）" zoomAndPan="'magnify' or 'dis 允许用户平移和缩放您的文件（默认Ma 层<svg>元素都需要安装SVG和它的命 xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xl

		xml:space="preserve"" 显现属性: All
switch		
symbol		
<text>	定义一个文本	x="列表的X -轴的位置。在文本中在第n个x轴。如果后面存在额外的字符，耗尽之后放置的位置。 0是默认" y="列表的Y x）0是默认" dx="在字符的长度列表中移志符号的绝对位置。（参考x）" dy="在移动相对最后绘制标志符号的绝对位置。rotate="一个旋转的列表。第n个旋转是符没有给出最后的旋转值" textLength="显示文本之间的间距/或字形调整的文本认：正常文本的长度）" lengthAdjust="指定长度就尝试进行调整用以呈现文本。是'spacing'和'spacingAndGlyphs'" 显现FillStroke, Graphics, FontSpecification, TextContentElements
textPath		
title	对 SVG 中的元素的纯文本描述 - 并不作为图形的一部分来显示。用户代理会将其显示为工具提示	
<tref>	引用任何SVG文档中的<text>元素和重用	相同的<TEXT>元素
<tspan>	元素等同于<text>，但可以在内部嵌套文本标记以及内部本身	Identical to the <text> element in additio 一个<TEXT>元素"
<use>	使用URI引用一个<G>,<svg>或其他具有一个唯一的ID属性和重复的图形元素。复制的是原始的元素，因此文件中的原始存在只是一个参考。原始影响到所有副本的任何改变。	x="克隆元素的左上角的x轴" y="克隆元素 width="克隆元素的宽度" height="克隆元 xlink:href="URI引用克隆元素" 显现属性
view		
vkern		

免责声明

W3School提供的内容仅用于培训。我们不保证内容的正确性。通过使用本站内容随之而来的风险与本站无关。W3School简体中文版的所有内容仅供测试，对任何法律问题及风险不承担任何责任。